Institutsbericht
IB 111-2008/07

Diplomarbeit

Behavior-based High Level Control of a VTOL UAV

Maurício Moraes Carneiro
Institut für Flugsystemtechnik
Braunschweig

164 Seiten
96 Abbildungen
48 Literaturstellen

Deutsches Zentrum für Luft- und Raumfahrt e.V.
Institut für Flugsystemtechnik
Abt. Systemautomation

Stufe der Zugänglichkeit: I, öffentlich und allgemein zugänglich.

Braunschweig, 18. Januar 2008

		Unterschriften:
Institutsdirektor:	Prof. Dr.-Ing. S. Levedag	_____
Abteilungsleiter (komm.):	Dr. R.V. Jategaonkar	_____
Betreuer:	M.Sc., Dipl.-Inf.(FH) F.-M. Adolf	_____
Verfasser:	Maurício Moraes Carneiro	_____

Dados Internacionais de Catalogação-na-Publicação (CIP)
Divisão de Informação e Documentação

Moraes Carneiro, Maurício
Behavior-based High Level Control for VTOL UAV ARTIS / Maurício Moraes Carneiro. São José dos Campos, 2008.
167f.

Trabalho de Graduação – Divisão de Engenharia Eletrônica – Instituto Tecnológico de Aeronáutica, 2008. Orientador: Prof. Dr. Karl Heinz Kienitz.

1. Behavior-based Control. 2. Intelligent Behaviors. 3. UAV High Level Control I. II. Comando-Geral de Tecnologia Aeroespacial. Instituto Tecnológico de Aeronáutica. Divisão de Engenharia Eletrônica. III. Behavior-based High Level Control for VTOL UAV ARTIS

REFERÊNCIA BIBLIOGRÁFICA

MORAES CARNEIRO, Maurício. **Behavior-based High Level Control for VTOL UAV ARTIS**. 2008. 167f. Trabalho de Conclusão de Curso. (Graduação) – Instituto Tecnológico de Aeronáutica, São José dos Campos.

CESSÃO DE DIREITOS

NOME DO AUTOR: Maurício Moraes Carneiro
TÍTULO DO TRABALHO: Behavior-based High Level Control of a VTOL UAV
TIPO DO TRABALHO/ANO: Graduação / 2008

_____

Maurício Moraes Carneiro
Rua Desembargador Carlos Xavier Paes Barreto, 380, ap. 1003, Mata da Praia
29065-330 Vitória – ES

BEHAVIOR-BASED HIGH LEVEL CONTROL FOR
VTOL UAV ARTIS

Essa publicação foi aceita como Relatório Final de Trabalho de Graduação

Maurício Carneiro
Maurício Moraes Carneiro
Autor

Prof. Dr Karl Kienitz (ITA)
Orientador

Dipl.Inf.(FH), M.Sc. Florian-Michael Adolf (Deutsches Zentrum für Luft- und Raumfahrt)

Prof. Dr Karl Kienitz Coordenador do Curso de Engenharia Eletrônica

São José dos Campos, 11 de novembro de 2008

RESUMO

Este trabalho contempla o problema de prover ao sistema embarcado de um veículo aéreo não-tripulado (VANT) autônomo as capacidades de planejar e conduzir missões complexas com auxílio externo reduzido e na ausência de módulos embarcados de planejamento de rotas e modelo de mundo digital. Tais capacidades são referidas neste trabalho como *Intelligent Behaviors* (comportamentos inteligentes). Três *intelligent behaviors* são propostos: *Fly Home* (voar para casa), *Search Object* (buscar objeto) e *Object Tracking* (perseguir objeto). O módulo *Fly Home* foi desenvolvido para prover ao sistema embarcado do VANT a capacidade de autonomamente retornar ao ponto de partida de uma missão através de uma trajetória segura. A abordagem proposta consistiu em reorganizar, adaptar e realizar as tarefas previamente executadas pelo VANT de forma a retornar pelo caminho originalmente percorrido. O *intelligent behavior Search Object* foi projetado para gerenciar uma missão de busca planejada *offline*, de forma a reconhecer as áreas em que a busca é permitida e lidar corretamente com possíveis interrupções. Para reconhecer as áreas de busca, a área total é dividida em células convexas e o algoritmo *Monotone Chain* é utilizado para o cálculo do perímetro de cada célula convexa. O *intelligent behavior Object Tracking* foi projetado para prover a capacidade de perseguir um objeto terrestre móvel com a assistência de dados providos por um sistema de reconhecimento visual embarcado. As características do movimento do objeto terrestre são estimadas utilizando um Filtro de Kalman. Este trabalho foi desenvolvido junto ao projeto ARTIS do Centro Aeroespacial Alemão (DLR), que tem como objetivo desenvolver sistemas para operar autonomamente helicópteros não-tripulados.

ABSTRACT

This work addresses the problem of providing the on-board system of an autonomous unmanned aerial vehicle (UAV) with the capabilities to plan and conduct complex missions with reduced off-board assistance. Such capabilities are referred to in this work as *Intelligent Behaviors*. In this work, three intelligent behaviors are proposed: *Fly Home*, *Search Object* and *Object Tracking*. The Fly Home intelligent behavior is intended to provide the UAV's on-board system with the capability of autonomously returning to the starting point of a given mission through a safe path. The behavior was designed considering the absence of online path planning and world model modules. The proposed approach is reorganizing, adapting and performing the tasks previously executed by the UAV so that the vehicle can return through the original path. The Search Object intelligent behavior is intended to manage a search mission planned offline, recognizing the areas in which the UAV is allowed to search and managing properly possible interruptions. To recognize the search areas, the total area is divided into convex cells and the Monotone Chain algorithm is used to calculate the perimeter of each individual convex cell. The Object Tracking intelligent behavior is intended to provide the capability of pursuing a moving ground object with the assistance of data provided by a visual recognition system. The motion characteristics of the ground object are estimated using a Kalman Filter. This work was developed with the ARTIS project at the German Aerospace Center (DLR), which aims to develop systems to autonomously operate rotorcraft vehicles.

FIGURES

NOMENCLATURE

3T	Three Tier Architecture
AI	Artificial Intelligence
ANN	Artificial Neural Networks
ARTIS	Autonomous Rotorcraft Testbed for Intelligent Systems
FT	Fly To
GCS	Ground Control Station
HITL	Hardware-in-the-Loop
HT	Hover Turn
HV	Hover To
IMM	Interacting Multiple Model
LD	Landing
NED	North East Down
pdf	Probability Density Function
PF	Pirouette Flight
PI	Pirouette Around XY
SD	Slow Down
SITL	Software-in-the-Loop
TO	Take Off
UAV	Unmanned Aerial Vehicle
UGV	Unmanned Ground Vehicle
UML	Unified Modeling Language
VC	Vision Computer
VTOL	Vertical Take-off and Landing
WT	Wait For
WO	Waypoint list off

SUMMARY

1 Introduction

The development of Vertical Take-Off and Landing Unmanned Aerial Vehicles (VTOL UAVs) has recently received special attention from the research community. Such aircrafts are adequate for general autonomous observation and reconnaissance tasks in different scenarios [1].

The Institute for Flight Systems at DLR (German Aerospace Center) has developed a project regarding the control and management of VTOL UAVs. The Autonomous Rotorcraft Testbed for Autonomous Systems (ARTIS) project aims the development of an inexpensive multi-purpose research platform based on commercially available helicopter models [1].

Figure 1: ARTIS rotorcraft

Even when performing autonomous missions, UAVs are normally assisted by human operators. There are, however, several situations in which it would be profitable to diminish the degree of dependability of the system with regard to human assistance. Consider the following scenarios depicted in figures 2 and 3.

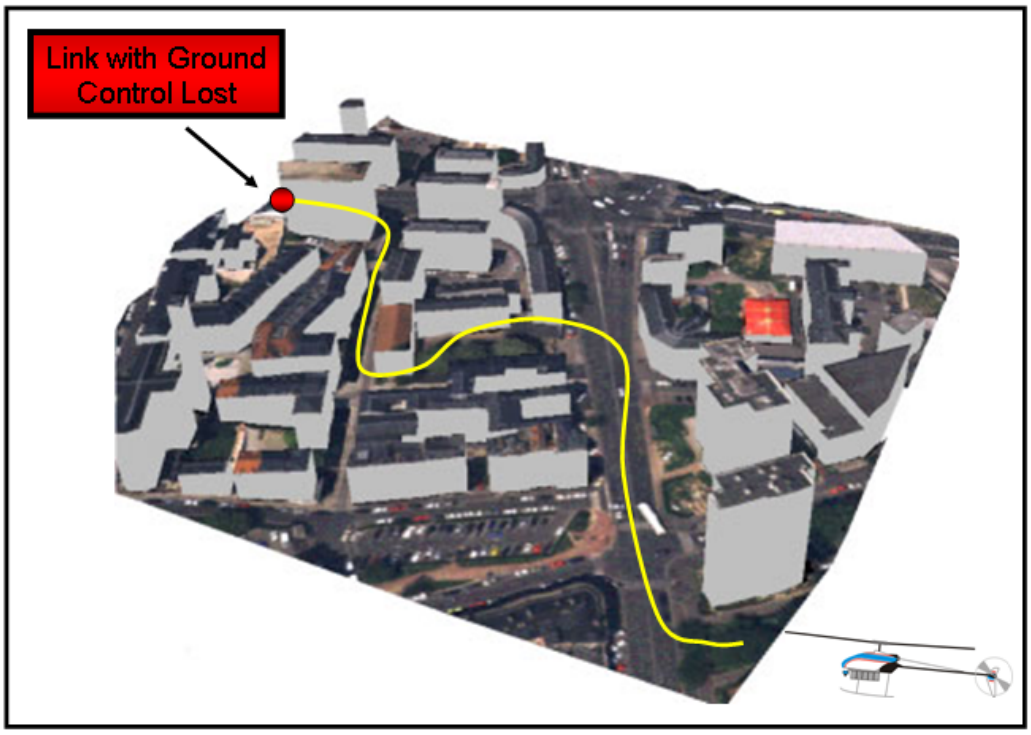

Figure 2: Urban mission scenario

In the scenario represented in figure 2, ARTIS executes a mission in an urban landscape. The UAV flies through the buildings following a mission planned offline by the human operator. At a certain stage of the mission, the UAV looses contact with the operator at the ground control station, waiting in vain for further instructions. In such circumstances, it would be desirable if ARTIS had the capability of returning safely to the mission's starting point.

In the scenario represented in figure 3, the UAV is commanded to execute a mission planned by the operator. This mission comprehends flying to a certain area sector of the Hannover Airport (designated in the figure), in which a ground object is searched. If the object is found, the aircraft is supposed to track it. The mission's last stage involves leaving the search area and reaching the mission's ending point as indicated.

In such scenario, several circumstantial aspects could interfere with the object tracking:

- The object could be moving away from camera view;
- The object could be briefly obscured or leave camera view;
- The sensors' data could (and generally is) noisy;
- The motion of the object could be errant and require fast motion response from the aircraft to keep visual contact;

The motion of the UAV could be too vigorous and cause the loss of visual contact with the object.

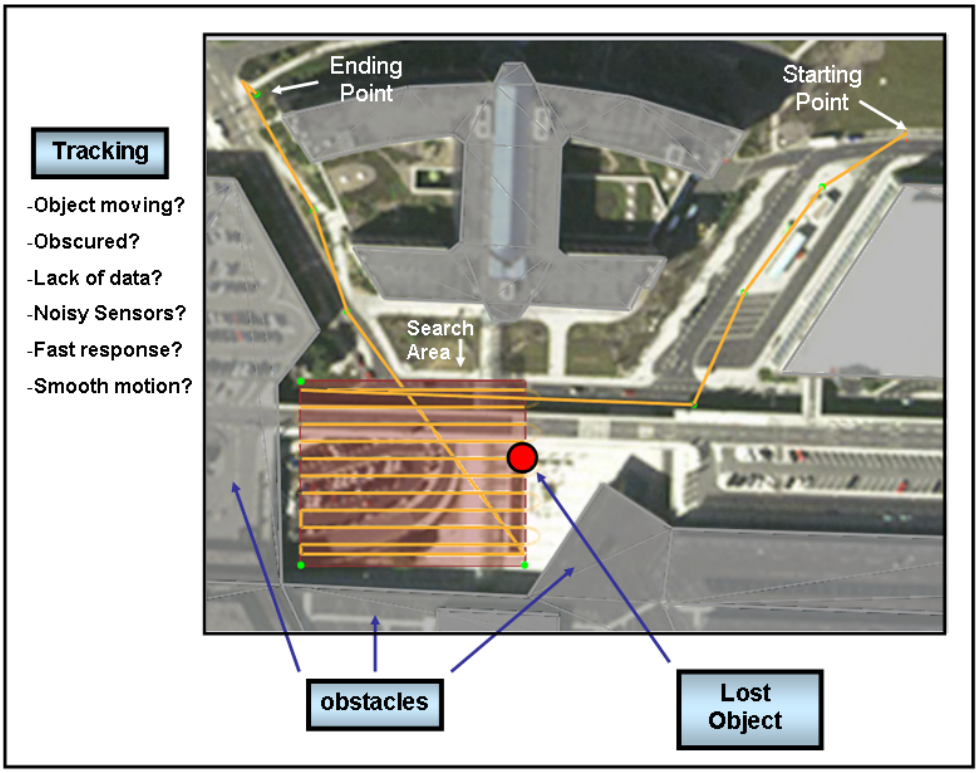

Figure 3: Search mission scenario

In the task of tracking the object, robustness against these practical difficulties might be essential in order to achieve success when pursuing the target.

Also, consider the search area shown in the picture. The area was obviously set in order to avoid the indicated obstacles. Therefore, it would be desirable if the system would restrain the tracking of the object within the search area, enhancing thus the safety of the mission.

Furthermore, consider that the aircraft has executed the mission until a certain point, when it found the ground object. The UAV then ceases the execution of the original mission and pursuits the ground object until it is lost. The mission is not yet accomplished, for one objective is reaching the ending point. Since the rotorcraft has moved out of the mission's path when tracking the object, the original mission cannot be resumed from the stage where it was interrupted. In such scenario, the UAV would have no alternative but to wait for direct commands from the human operator.

Such scenarios present evidence on the advantages of enhanced autonomy and intelligent behavior for the UAV. Normally, such upgrade could be reached by making use of

onboard planning and recognition modules, such as a digital world model, a path planner and a collision avoidance system. However, these modules are complex and require great online computation capability.

An alternative to using such complex modules would be using the mission planned offline and use simple assumptions in order to allow online planning of safe paths and missions, resulting in an overall more intelligent behavior. These assumptions are intuitive and valid for most of the situations, and include assuming that the original mission's path is safe, that the inside of a search area is free of obstacles and it is possible to return to the mission's path and restarting it's execution from the stage relative to such position.

This work addresses the problem of enhancing ARTIS' autonomy in the absence of path planning and world model modules. For such purposes, we present the development of three modules, which we refer to as *Intelligent Behaviors*. The intelligent behaviors are modules which actuate in ARTIS' high level control layer, providing high level control organized in sequences of tasks or waypoints.

This thesis is organized as follows: in Chapter 2 we present the work related to the topic of providing UAVs with intelligent decision-making capabilities. In Chapter 3, we present aspects of the ARTIS' system relevant to this work, providing the necessary knowledge for understanding the development of the work. In Chapter 4, we address the intelligent behavior *Fly Home*. In Chapter 5, we address the intelligent behavior *Search Object*. In Chapter 6, we address the intelligent behavior *Object Tracking*. In Chapter 7, we present considerations for future work. In the chapters concerning the intelligent behaviors, we describe the problem addressed in each case, the related relevant works, the approach and results obtained with each module.

2 Related Work

The research community has witnessed a growing interest and development in the area of unmanned aerial vehicles. The quest for the improvement of UAV autonomy has been one of the main related topics, and several artificial intelligence (AI) techniques have been applied in order to enhance UAV systems' decision-taking and planning capabilities.

One alternative proposed by classical AI is using knowledge-based systems aiming to reproduce human-like cognitive capabilities which would provide the autonomous agent with high-level control [2] [3]. Due to the similarities between the system and folk-psychological reasoning, the implementation is simplified and intuitive. However, such achievement comes at the cost of run-time complexity, a crucial factor in embedded systems with limited computational power [4]. Cognitive systems also have the disadvantage regarding real-time reflexive actions (i.e. a direct connection between sensor and actuator), which invoke a high variance of unpredictable AI techniques. Another problematic aspect of such approach concerns the coordination and mediation of reflexive behaviors with the system's overall behavior. When a reflexive behavior overcomes the normal deliberation mechanisms, the system might be ineffective to reason about and affect the real-time reaction. Moreover, it is not yet clear to which extent do humans use deliberation and reasoning to take decisions [5]. This fact directly defies the premises of reproducing human-like decision making processes.

Another approach would be behavior-based control with the Subsumption Architecture [6] [7]. Systems using such paradigm do not necessarily seek to produce cognition, relying instead in a hierarchy of reactive behaviors designed for a specific purpose. In such systems, behaviors with a higher hierarchy status directly influence those of lower hierarchy. A common approach consists into superposing the influence of elementary behaviors, resulting in a new, emergent behavior. In this way, the behaviors are the building blocks of the system, and the functionality is emergent [8]. The goal of many behavior-based approaches is use machine learning to enable the system to "learn" configurations of behaviors which leads to achieving the agent's goals. From a run-time complexity point of view, behavior-based approaches have shown good efficiency. One of the disadvantages of a number of behavior-based approaches is the fact that the interconnection of behaviors results in a system difficult to be explained [9]. Furthermore, optimality in such approaches is hard to be achieved [10].

Cognitive modeling has been used in UAV projects to provide the decision-making control capabilities necessary to improve the autonomy of the vehicle [4]. The approach presents the advantage of mapping intuitively to everyday notions those constructs that are used by pilots to describe their mission control decision-making. It is also argued that such an approach has the advantage over behavior-based approaches of being more extensible, since the later has a "purpose-built" design [4].

Behavior-based approaches have also been used in the control of UAVs, using individual behavior modules whose effects are superposed, building a complex behavior on the top of lower-level ones [11]. However, such approach has been applied for *navigation control*, not being applied to planning or decision-taking.

Another approach frequently used by UAV projects is of separating the vehicle's control into layers, including a deliberative layer and a reactive layer [12] [13]. In such systems, the reactive layer contains reactive programs responsible for executing lower level tasks. The deliberative layer on the other hand comprises a planner that reasons about goals, resources and timing constraints with well known AI techniques [14].

Pondering on the advantages and disadvantages of each approach, a solution for providing intelligent behavior to ARTIS has been proposed. A deliberative layer was chosen to support and manage modules designed for intelligent, high-level control of the vehicle. Such modules were inspired by behavior-based paradigm, being designed in a purpose-directed fashion using simple algorithms. This approach avoids the computational disadvantages of cognitive models without suffering of lack of extensibility, since the deliberative layer can accommodate behavior modules with distinct characteristics. In other words, the behaviors are purpose-built but the deliberative layer is not, providing hence possibility of extensions. The complexity of analysis of emerging behaviors encountered in behavior-based approaches is suppressed by the use of the deliberative layer to manage the intelligent behavior modules, maintaining no more than one active. Moreover, the intelligent behavior modules in the deliberative layer make use of the reactive behaviors in the reactive layer, planning task-based missions. This way, the planning and system management regarding high-level objectives is decoupled from the execution of the individual reactive tasks.

Another common practice in the development of intelligent UAVs is the use of digital world model and path-planning onboard modules, being used by a number of UAV projects to increase the vehicle's autonomy [15] [16]. With such modules, the UAV's embedded system uses a digital mapping of the environment's obstacles and a path-planning algorithm to plan

safe paths connecting desired starting and ending points. By providing the ability of autonomously calculating safe paths and the spatial awareness of the presence of obstacles, these modules enhance the vehicle's autonomy.

However, such contribution comes at the cost of complex implementation issues and high computation demands. The development of the intelligent behaviors for ARTIS is intended to introduce a different perspective in the enhancement of the vehicle's autonomy. Using simple assumptions (e.g. the path planned offline by an operator is obstacle-free, the area within a search perimeter does not contain obstacles), we propose intelligent behaviors with planning capabilities which regard high-level commands (e.g. fly home, track object) without the necessity of world model and path-planning modules. Although the intelligent behaviors do not present an alternative to such modules, they introduce planning and management capabilities which take account of the mission's goals (e.g. returning home), enabling autonomous operation and decision-taking in a higher level.

3 ARTIS System Remarks

As stated before, the work presented in this document was realized within the scope of ARTIS' project. Therefore, in order to understand the problems faced and solutions proposed in the project, it is necessary to acquire some previous knowledge about the system behind ARTIS.

The ARTIS system comprehends a wide range of specific modules addressing different necessities, such as avionics hardware and software, visual processing, flight control, mission management, mission planning, digital world modeling, simulation environments, etc. In this chapter, we introduce and explain aspects of the system considered relevant to the scope of this work.

3.1 Mission Management

The core of ARTIS' embedded system includes modules with different responsibilities, such as the navigation, flight control and mission management. The scope of this work concerns the Mission Manager module, which addresses the high level control of the UAV. The following subsections present aspects of the Mission Manager whose acquaintance is necessary in order to understand the development of the work presented in this document.

3.1.1 Sequence Control and Supervisor

The framework of the Mission Manager is defined by a 3T (Three Tier) Architecture [14] in order to handle system complexity while being flexible and fast enough for updates and extensions [9]. The architecture is organized into two layers, namely *Sequence Control* and *Supervisor*. Figure 4 presents such architecture.

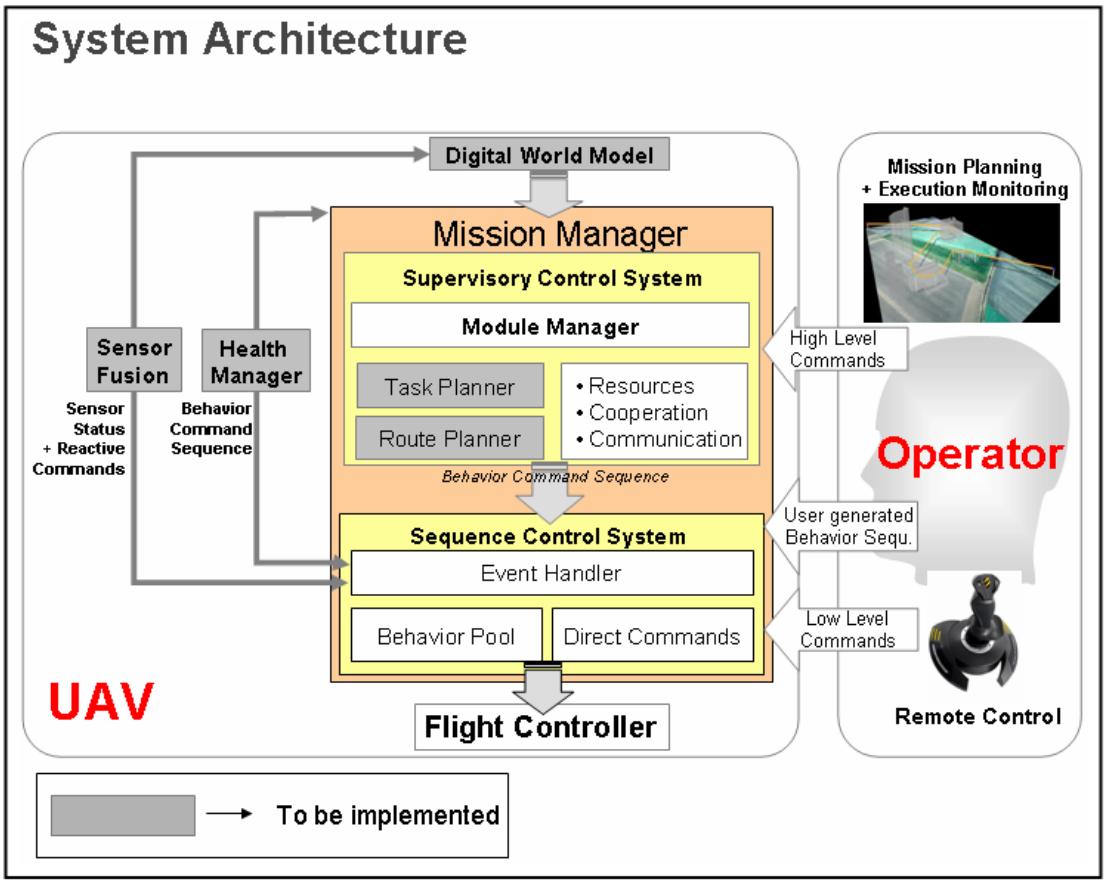

Figure 4: Mission Manager Architecture

The Sequence Control module constitutes the executive part of the architecture, executing commands generated by mission consisting in sequentially organized tasks and commands directly sent by the operator. The Sequence Control System's decision logic is modeled using Unified Modeling Language (UML) Statechart diagrams [17] in order to provide safe coordination between multiple events [9].

Generally described, the Supervisor module is a deliberative layer responsible for taking high level decisions based on internal and external events. At every cycle which the system operates, this layer is accessed before the Sequence Control layer processes a specific task. This allows the Supervisor to modify a task or a mission when conditions are recognized to imply necessity of modification.

The Supervisor is responsible for managing requests from the ground operator, as well as events such as the loss of contact with the GCS. Furthermore, the Supervisor is the entity in charge of managing the request, management and execution of the intelligent behaviors developed in this work. Therefore, the Supervisor retains planning capabilities, and recognizes the associated high level objectives, e.g. Fly Home.

The Supervisor has been modeled as a Statechart diagram according to the specifications of UML. Several favorable aspects of statechart modeling corroborate to this choice. Statecharts have been thoroughly studied, resulting in abstract testing techniques which allow (semi-) automatic verification of the model. Furthermore, there exists good software tool support which significantly facilitates development and implementation [9]. Statechart-based modeling has also been successfully used to address the high-level control of VTOL UAVs [18].

Figure 5 illustrates the statechart model for the Supervisor module.

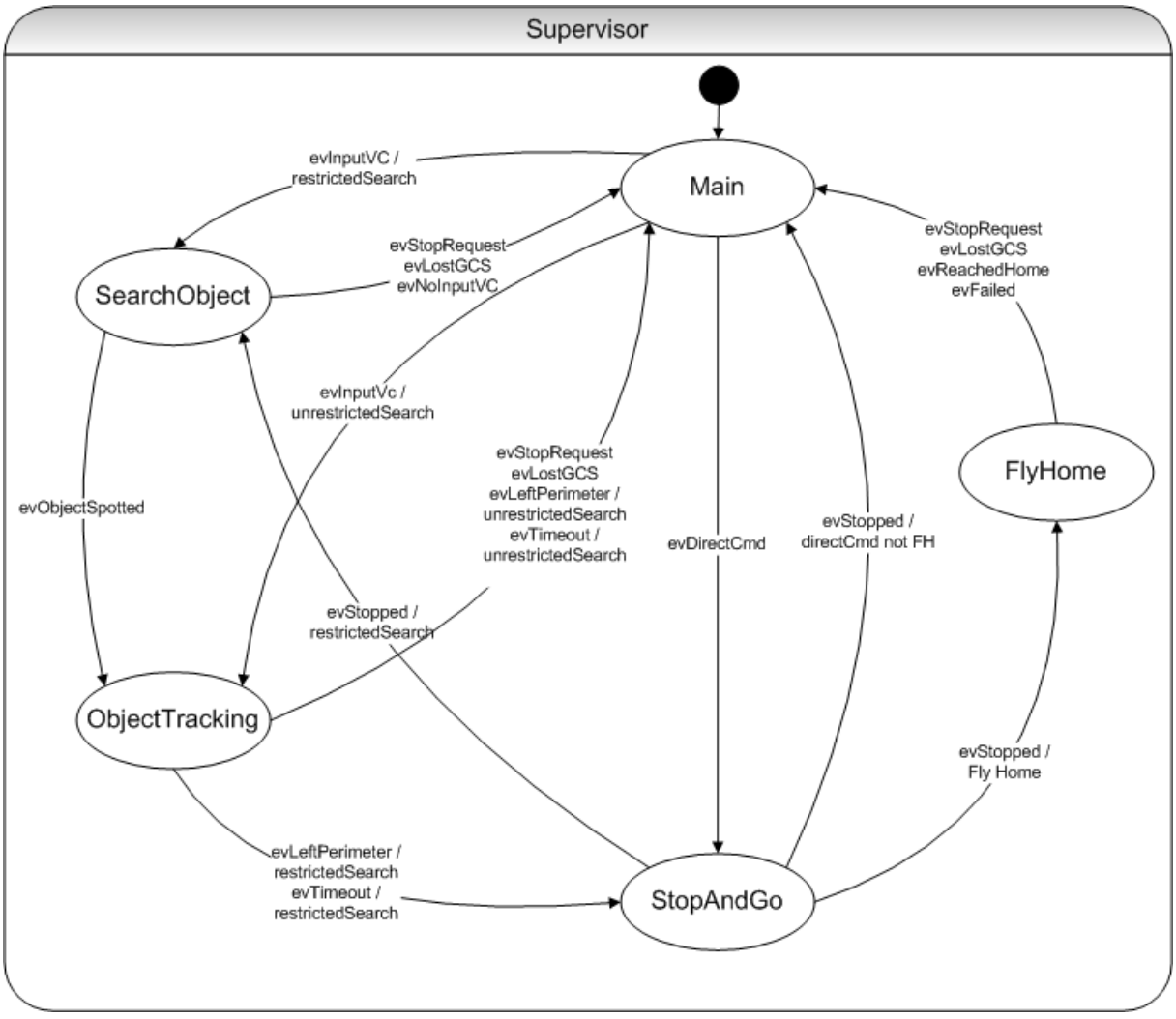

Figure 5: Supervisor statechart model

The Supervisor is organized in five different states. The transition between states is triggered by events which in some cases have associated guard conditions. The states *Search Object*, *Object Tracking* and *Fly Home* are associated with intelligent behaviors and were designed to provide the proper management for these behaviors. The state *Main* comprises

non-specific missions, as well as operation modes not contemplated by the other states of the Supervisor. Situations operating in Main state include regular (i.e. not influenced by intelligent behaviors) autonomous missions, direct position commands (i.e. task commands issued directly by the operator) and manual (remote control or joystick) control. The state designated *Stop and Go* assures safe transitions from operating modes, including the execution of direct commands and the passage to states Search Object and Fly Home. In both cases, before executing the direct command or entering the state associated with the requested intelligent behavior the UAV is ordered to cease the original mission, slow down and stop before receiving permission to perform the requested intelligent behavior.

The development of several aspects of the Supervisor module was also part of the scope of this work. These improvements are presented and discussed in chapters 3, 4 and 5.

3.1.2 Behaviors

In the ARTIS' embedded system, the basic physic capabilities (e.g. hovering to point *P* while maintaining heading *H*) are organized in skill modules. These reactive skills are behaviors (control laws) tightly coupled with environment through sensor readings and actuators. These skills shall be referred as *Behaviors* and were designed following the *Behavior-based paradigm* [19], [20]. Behaviors make "simple-world" assumptions such as, the sensor input is valid and desired goal can be achieved [9].

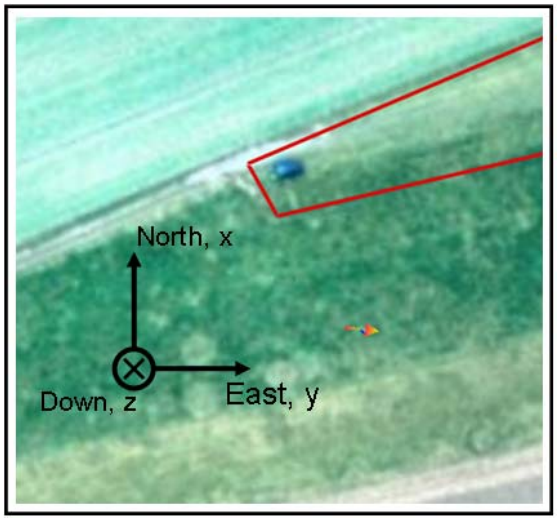

Figure 6: NED system

The behaviors take account of a geographical coordinate system following NED (North East Down) convention with respect to a geographical reference defined by the operator. For convenience, we will refer to North coordinate as *x coordinate*, East coordinate as *y coordinate* and Down coordinate as *z coordinate*. Also, the convention for angular displacement used is that positive angles follow clockwise direction and negative angles follow anti-clockwise direction. Figure 6 presents a graphical example of a NED reference system.

Table 1 presents the different types of behaviors, with a brief explanation of their specific goals and parameters.

Table 1: Behaviors

Behavior Type (Abbreviation)	Objective	Parameters
Hover Turn (HT)	Maintain geographic position while rotating the heading of the helicopter	Angular displacement [°] Angular speed [°/s]
Wait For (WT)	Maintain geographic position and heading for a time period	Waiting time [s]
Slow Down (SD)	Slow down the UAV until reaching null translation and angular velocities	None
Take Off (TO)	Take off flight while maintaining x, y coordinates	Height [m] (optional)
Landing (LD)	Land while maintaining x, y coordinates	None
Pirouette Flight (PF)	Performs an “acrobatic” flight, rotating while translating	Destination x, y, z [m] Turning speed [°/s]
Pirouette Around XY (PI)	Perform a radial trajectory while maintaining initial relative heading to center	Center coord. x, y [m] Angular speed [°/s] Angular displacement [°]
Hover to (HV)	Hover to a destination with a specific heading	Destination x, y, z [m] Heading [°]
Fly to (FT)	Perform a fast flight following a trajectory planned as a spline [20]	Destination x, y, z [m] Time [s] Spline parameters dx, dy, dz

As shown on the table, some behaviors are related to the UAV's capability of following a certain path in order to reach an ending point. Figures 7, 8 and 9 illustrate typical trajectories executed during some of these behaviors. The figures were obtained using Software-in-the-Loop (SITL) simulation (subsection **3.2.2**).

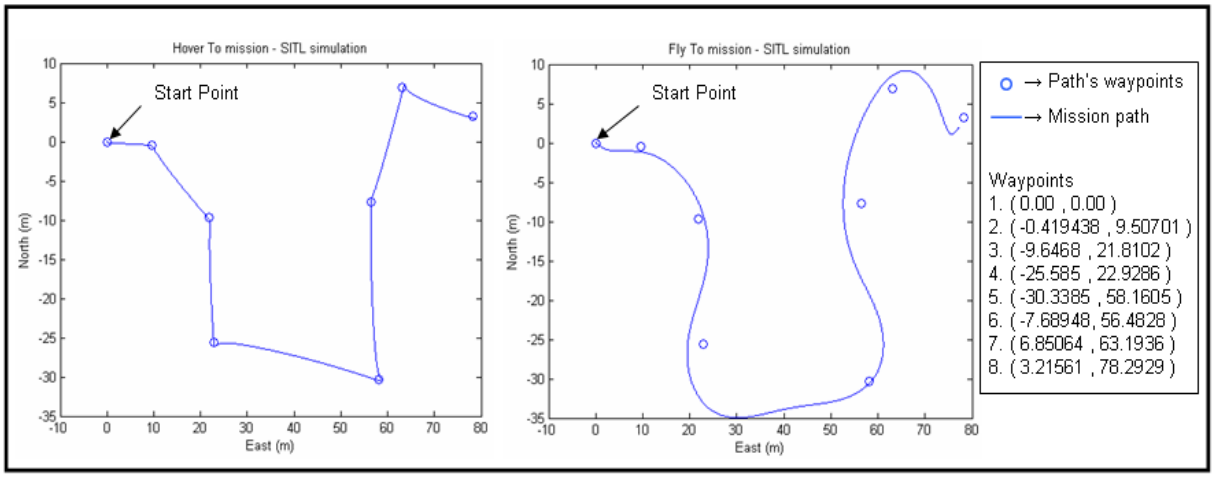

Figure 7: SITL simulation, Hover To and Fly To behaviors

In figure 7, we show the difference between a mission using Hover To behaviors and one using Fly To behaviors to cover a set of waypoints. In both missions, the waypoints to be visited are the same. On the left side of the figure, the path was covered using Hover To behaviors. As seen, the path segments are "almost" straight. The UAV stops at each waypoint, turns and flies directly to the next waypoint. On the right side, the path was covered using Fly To behaviors. The UAV follows the trajectory without stopping at the waypoints.

Figure 8 depicts a Pirouette Flight behavior executed in SITL simulation. In this mission, the UAV was supposed to fly from point (0, 0, -5) [m] to point (0, -40, -5) [m]. The flight is intended to demonstrate the aircraft's control capabilities, with the helicopter spinning around while flying. On the left side, we see the path of the task. On the right side, we see the evolution of the helicopter's heading through the mission.

Figure 9 depicts a Pirouette Around XY behavior executed in SITL simulation. In the simulation, the aircraft was supposed to follow a radial trajectory around point (0, 30) [m] for an angular displacement of 270° while keeping the initial heading relative to the center. This type of pirouette is used often in missions in which ARTIS is supposed to fly around an object and take pictures with different perspectives of the object.

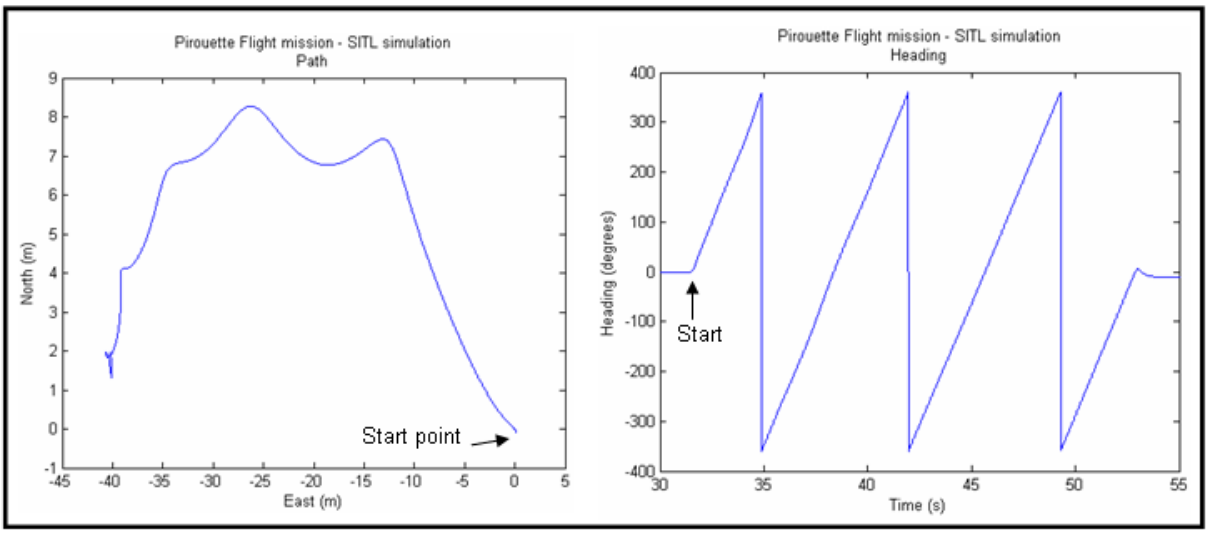

Figure 8: SITL simulation, Pirouette Flight behavior

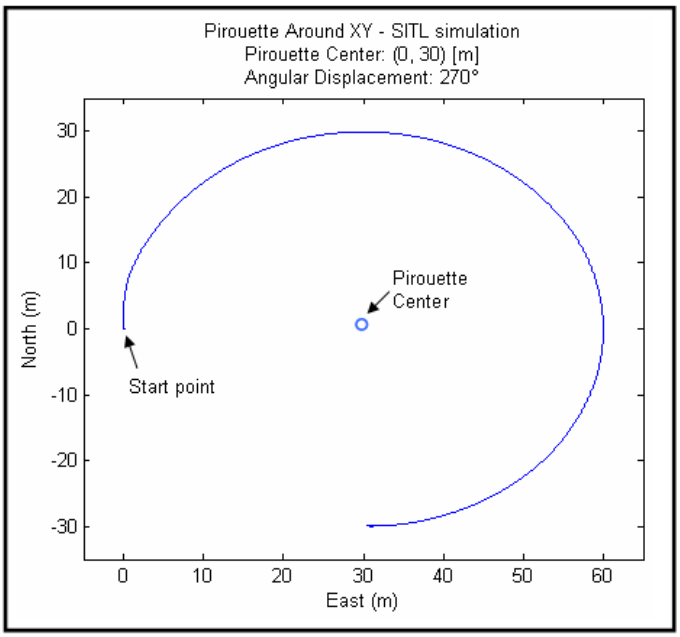

Figure 9: SITL simulation, Pirouette Around XY behavior

3.1.3 Behavior Sequence

The *Behavior Sequence* is an entity which describes the tasks of a mission and the order in which they should be executed. Behavior sequences are organized as arrays of behaviors indexed properly according to precedence of execution. At any given time during a mission an internal cursor in the behavior sequence enables access to the behavior scheduled for the next task.

A behavior sequence can be represented by a list named *Waypoint list*, which describe from first to last the tasks on a mission. Each task is represented by its name or abbreviation and parameters. Figure 10 presents an example of waypoint list.

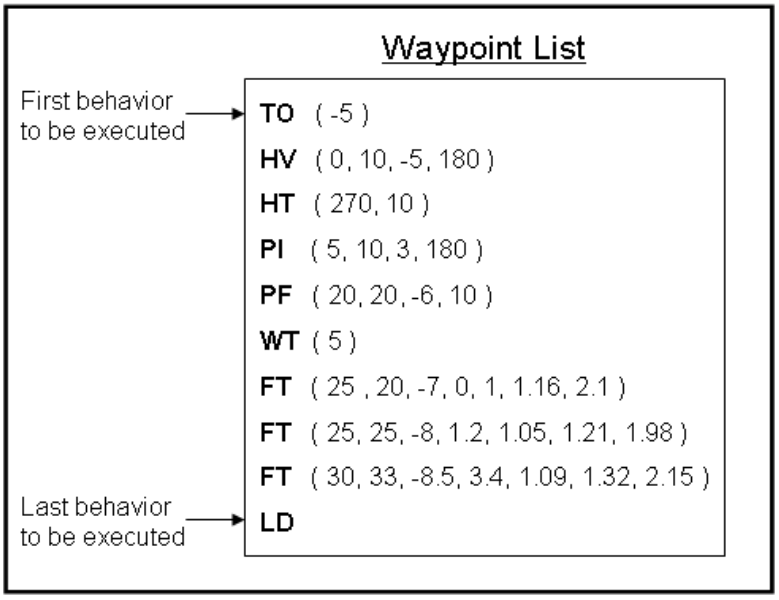

Figure 10: Waypoint list example

3.2 Other Aspects

In this section, we present other relevant aspects of the ARTIS project. These aspects do not refer to ARTIS' embedded system.

3.2.1 Ground Control Station and Maestro

The ARTIS project makes use of a complete computer platform for mission control by the operator, named *Ground Control Station* (GCS). The GCS is used to run a software developed by the ARTIS project named *Maestro*. This software enables the communication with the UAV via a wireless data link. The abilities of Maestro include onboard sensors data reading and logging, access to the aircraft's subsystems' status, voice and joystick direct commands, display of camera images, flexible mission planning, re-planning and monitoring.

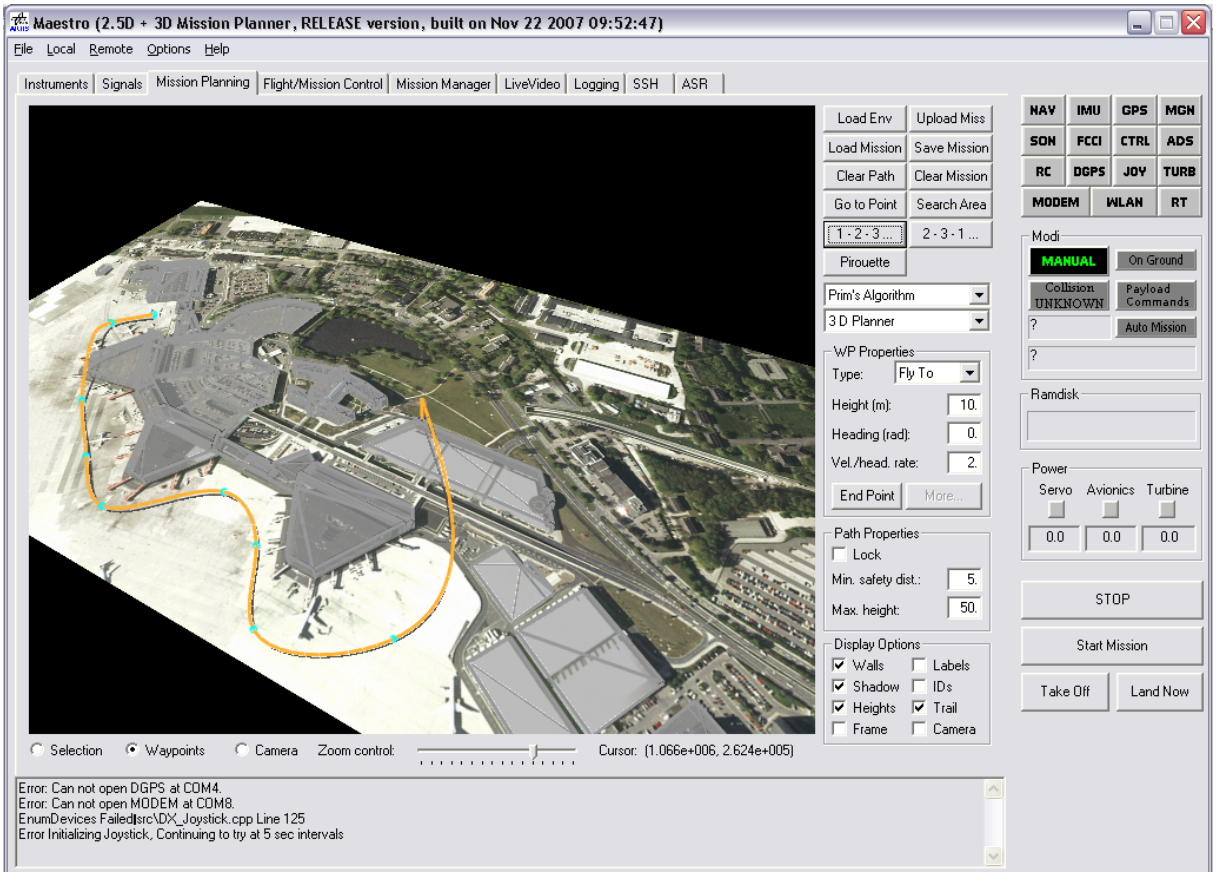

Figure 11: Maestro software's mission planning interface

3.2.2 Software-in-the-Loop (SITL) Simulation

As an alternative to testing ARTIS' system in a real flight test, simulation setups provide the project with the possibility of preliminarily testing and tuning new algorithms and features proposed. One of such setups developed by the ARTIS project is the *Software-in-the-Loop* (SITL). This setup constitutes a 6-dof dynamic model of the aircraft implemented in Simulink environment, with models of ARTIS' sensors (including noise and disturbance), actuators (including dynamic limitations) and the navigation filter used by the helicopter.

3.2.3 Hardware-in-the-Loop (HITL) Simulation

The *Hardware-in-the-Loop* (HITL) simulation environment consists into an enhancement of the SITL setup, using a dedicated simulation computer and providing a more realistic and complete setup. The system's dynamics is simulated online using a Simulink dynamic model. The inputs of the setup consist into actuator commands generated by the UAV's flight control computer. The outputs of the setup are emulated sensor signals. In such

environment, ARTIS' flight control computer interacts with an artificial, simulated environment, operating as if it were controlling the aircraft.

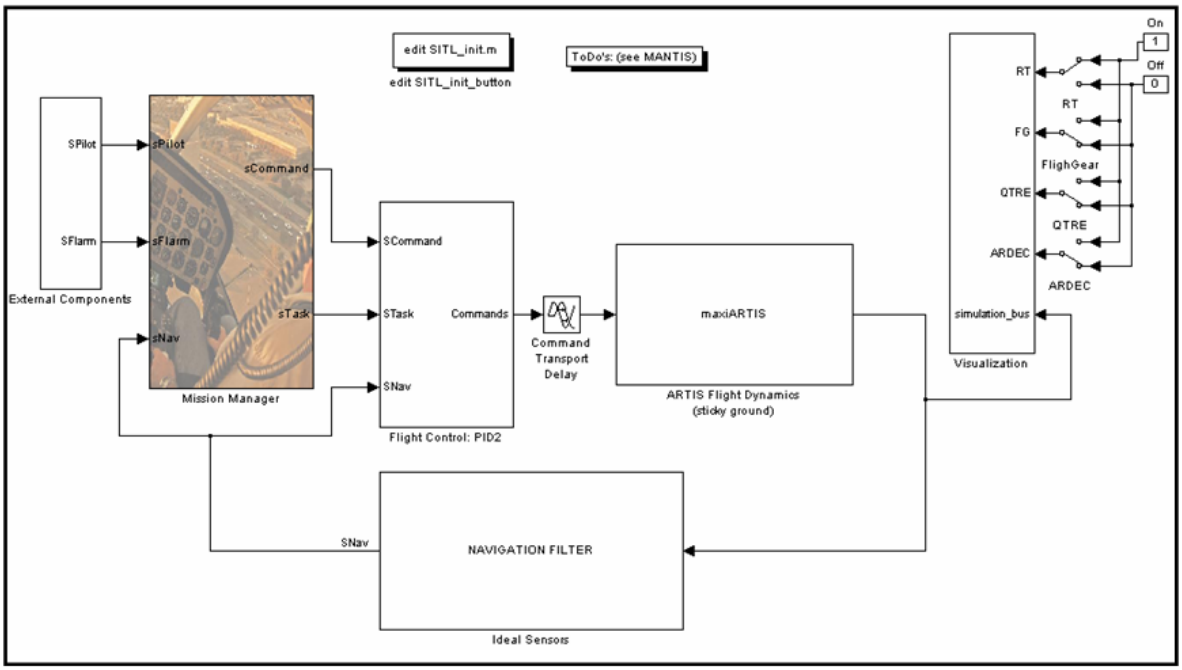

Figure 12: SITL's Simulink layout

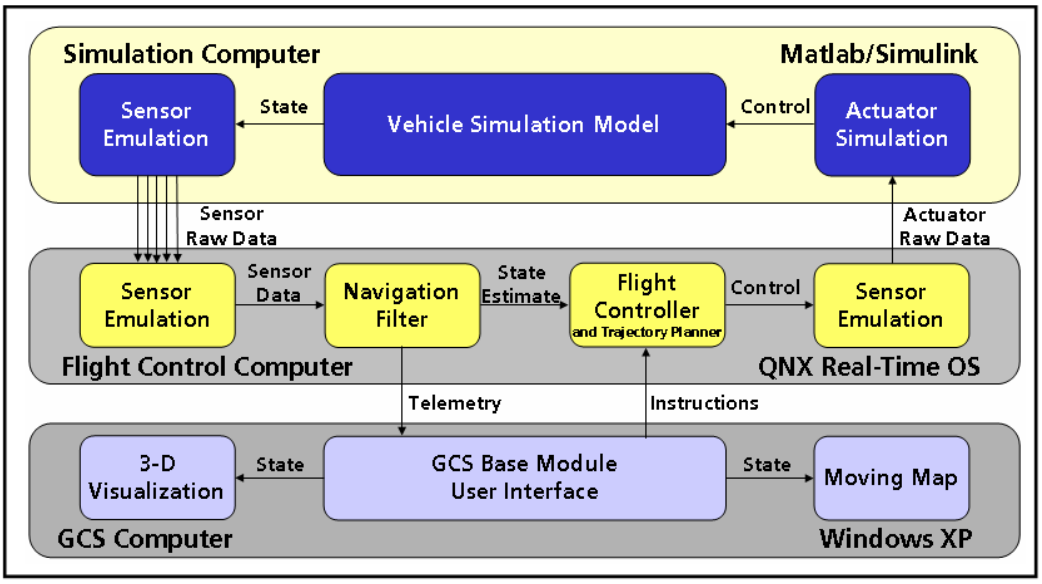

Figure 13: HITL layout

Relying on such features, the HITL simulation setup allows reliability tests of all software modules and performance and efficiency evaluations for the control and navigation systems. The flight data (including position, velocity and other sensor information) is sent to

the GCS and to an OpenGL-based graphic display, which provides a visual output of the mission's evolution.

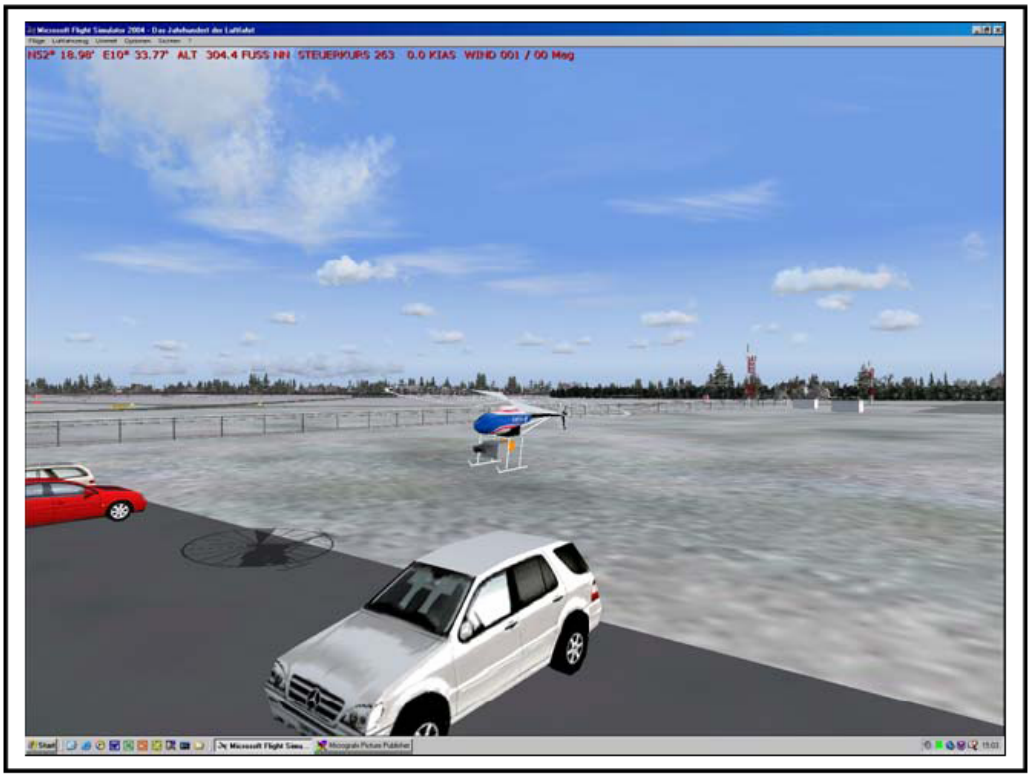

Figure 14: HITL simulation visual output

4 Fly Home

This chapter describes the intelligent behavior module designated *Fly Home*. This module has been the first of the intelligent behaviors to be conceived, implemented and successfully tested. The main objective of the Fly Home behavior is to empower the system with the capability to return to the starting point of a mission without off-board assistance. Similar features have also been developed by commercial[†] VTOL UAV systems.

Fly Home was designed to operate without sufficient information about its environment or planning capabilities (i.e. digital world model or path planning modules) which would allow the on-board system to directly plan a path to a position designated Home. The task planning in such conditions is possible due to the organization of the missions into sequences of timely organized basic tasks described in section **3.1**.

Since the intelligent behaviors are intended to enhance the system's autonomy, it is important to define the concept of *autonomy*. In the context of the study of unmanned systems, a possible definition would be that autonomy is the system's capabilities to achieve its mission goals. Therefore, the level of autonomy of a system can be related to the complexity of the goals it is able reaching [22].

The Fly Home problem involves solving situational questions (e.g. where is Home?) using simple assumptions (e.g. the original mission is safe). The mission behavior resolution is more complex in this case than simply following a set of waypoints leading Home planned by the operator, which constitutes in an enhancement of autonomy of the system [18].

Furthermore, the mission Fly Home is planned and executed on-board upon request from the operator. Normal missions, i.e. missions which are not associated with intelligent behaviors, have to be planned offline by the operator. As the operator's involvement with the planning of the Fly Home mission is resumed with the pressing of a button, the Fly Home behavior enhances the autonomy of the system [23].

This chapter defines formally the concept of Home in the scope of the Fly Home behavior, establishes the necessary conditions to guarantee the possibility of computing a safe mission leading the UAV back Home and presents an algorithm to calculate this mission. Furthermore, the modeling of the Fly Home behavior using a statechart is presented as well as strategies used by the behavior to manage events and state transitions from the moment when

[†] http://www.swiss-uav.com/swiss-uav/products/bap-deladobbrautopilot.html

a Fly Home request is received by the on-board system until the moment when the system returns to its normal operation state.

4.1 Problem Statement

The Fly Home behavior addresses the problem of making it possible for the UAV to interrupt a mission in the course of its progress and return to the starting location (designated *Home*) of the mission through a safe collision-free path. The only assumption made is that the original mission provided by the operator provides a safe collision-free path.

This behavior enables the safe autonomous execution of a high-level task (flying home) upon online request without necessity of off-board assistance or offline planning. Thus, the Fly Home behavior enhances the autonomy of the ARTIS system in the absence of on-board path-planner or world model.

Although the presence of path planning or world model modules is not necessary, the Fly Home behavior is intended to make use of such features when their availability is granted. In such missions, the behavior must be able to define a Home position, which could be relative to the current mission or a specific location pre-defined by the operator.

Considering the aspects debated above, a set of requirements with which the solution must comply have been established.

1. The re-planning must be done online.
2. No on-board path-planner or world model should be necessary.
3. No additional history-keeping module should be required from the current system, i.e. there must be no necessity to record visited waypoints or additional information about the original mission.
4. The only information required from the operator must be a Fly Home request, i.e. the planning and execution of the new mission must not require further information or supervision from the operator.
5. The resulting path must not contain points which safety status is unknown by the on-board system. In the presence of onboard world model and path-planner, this condition is respected This requirement could be translated in the following form:

Another aspect of the problem addressed by Fly Home is that the objective of “flying home” requires a contextual analysis of what home is: each mission could have a home location, or home could be simply a position pre-defined by the operator.

Additionally to computing the Fly Home mission, the behavior module must also provide the management of the task from beginning (i.e. the reception of a Fly Home request) until end (i.e. when Home is reached or when the task has been interrupted). The management mentioned includes safely halting the original mission, planning the mission towards home, handling failures or external events relevant to the mission (e.g. impossibility of calculating mission towards home, stop request from GCS) and safely returning to the normal operations mode.

4.2 Discussion

The assistance of onboard path planner and world model modules implies that, as long as a safe path between two points exists, this path can be calculated online [24]. The path generated is compliant with requirements 1, 3, 4 and 5 presented on section **4.1**. Therefore, the mission leading the UAV from the position where a Fly Home request was issued to the position considered to be home can be simply calculated by the path planner. The key issue in such case would be developing the mission management described on section **4.1**.

In order to comply with requirement 2, it is necessary to provide the behavior module with the capability to compute the mission conducting the UAV to home without the assistance of on-board path-planner or world model modules. The absence of such modules implies that the on-board system of the UAV ignores physical characteristics of its environment such as the presence or absence of obstacles in most given points. The only points assumed to be obstacle free are the ones contained in the path of the original mission, as described by the assumption that the original mission provided by the operator provides a safe collision-free path (section **4.1**). In such conditions, the requirement 5 can be formally expressed by the expressions in (4-1).

Let $P_{FlyHome} \subset \mathbf{R}^3$ and $P_{Mission} \subset \mathbf{R}^3$ be sets of points. (4-1)
Let $P_{Mission}$ represent the set of points contained by the intended path of a mission M.
Let $P_{FlyHome}$ represent the set of points contained by the intended path of a Fly Home mission relative to M.
$P_{FlyHome}$ is a valid set of points of the path described in the Fly Home mission relative to $M \Rightarrow P_{FlyHome} \subset P_{Mission}$

The on-board system retains information about the original mission through a behavior sequence, which is calculated online using a waypoint list provided by GCS. The information contained on behavior sequence is useful to partially calculate the path followed by the UAV during the course of the mission.

The approach proposed for the problem would be to assemble a new behavior sequence, and therefore a new mission, which would reach the point designated as "home" while following the calculated original mission path, assumed to be safe and collision-free. In other words, the UAV would fly the "inverse path", executing and adaptation of the original tasks in the inverse order. The proposed approach to compute the mission achieves the objectives of the Fly Home behavior while respecting all the restrictions imposed.

The computation of a new mission leading to home, however, does not constitute the whole solution. It is also necessary to provide the behavior with the mission management capabilities described before.

The approach proposed to develop the Fly Home behavior's management capabilities is to model the behavior using Statechart diagrams. Such model is useful for our purposes since it provides tools to handle external events and the process' internal state transitions.

Hence, the development of the solution was parsed into two parts: computing the Fly Home mission and managing the Fly Home mission.

4.3 Computing the Fly Home Mission

As stated before, in the presence of on-board path planner and world model modules the mission leading the UAV to Home position can be calculated simply by using the path planner. Therefore, the problem of computing the Fly Home mission is relevant for the design of the behavior only in the absence of such modules. To compute a new mission towards home in such circumstances, it is first necessary to understand how would one infer the original mission's path from the behavior sequence and how do each behavior in the sequence affects this path.

4.3.1 Computing the Original Mission Path Waypoints

Some behaviors cause the UAV to displace its geographical position. These are, therefore, the behaviors which allow the pathway to be calculated. We will forth designate this set of behaviors *Position Behaviors*, and the complementary set of behaviors will be

designated *Non-Position Behaviors*. The following table presents the components of each set of behaviors:

Table 2: Position and Non-Position behaviors

Position Behaviors	Non-Position Behaviors
Hover to	Hover turn
Fly to	Slow down
Pirouette around XY	Wait for
Pirouette flight	Landing
	Take off

Although the landing and takeoff do cause geographical displacement of the UAV, they are considered position behaviors in the Fly Home context. This is due to the fact that, during a regular mission, if the UAV at some point lands, takes off and continues the mission, the requirements of the Fly Home mission allow the behavior to follow the original path without the necessity of landing, taking off and continuing. In other words, the behaviors Take Off and Landing are not relevant when calculating the path through which the UAV must fly during a Fly Home mission.

Forth, we use terms such as *indicate*, *designate* and *define* a waypoint to establish the ending point of a trajectory performed during a position behavior task. For example, the waypoint designated by a behavior Hover To with destination to point X is the point X itself. Likewise, the waypoint defined by a behavior of type Pirouette Around XY is the last point of the trajectory followed during the task.

Also, the starting point of a task will be classified as *intended* and *actual*. The intended starting point of a position behavior indicates the waypoint designated by the first precedent position behavior. The actual starting point of a position behavior indicates the position of the UAV at the moment when the respective task begins.

As formerly stated, it is possible to partially calculate the path covered by the UAV during a mission by the behavior sequence. The complete path cannot be inferred since the behaviors from the behavior sequence do not define path segments. All the position behaviors, with the exception of Pirouette around XY, define only one waypoint (the destination) to be visited with their parameters. Pirouette around XY defines with its parameters only the center and the angular displacement of its radial trajectory. However,

when analyzing the position behaviors sequentially it is possible to calculate the waypoint designated by the end point of the pirouette around XY, as shown by the following function.

The pseudo-code of function 1 describes a method to calculate the waypoints of a mission.

```
Function 1:     computeWaypoints( behaviorSequence, waypoints )
     define:
     index iter ← 0;
     for each behavior behaviorSequence[i]
          if behaviorSequence[i] is of type Pirouette Around XY
               waypoints[iter] ← end point of arc with starting point at
               waypoints[iter-1], center at behaviorSequence[i].center and arc angle
               of behaviorSequence[i].angularDisplacement ;
               iter ← iter + 1;
          else if behaviorSequence[i] is a Position Behavior
               waypoints[iter] ← behaviorSequence[i].destination;
               iter ← iter + 1;
          end if
     end for
```

4.3.2 Defining Home

Before we define a method for calculating a mission leading to home, it is necessary to define where precisely is *home*. The concept of home should consider both the behavior objective – reaching the mission's starting point – and the restrictions imposed on the solution. The behavior was also designed so that the operator could choose a

In the behavior sequence, there is the possibility that one or more behaviors indicate an interruption of the autonomous mission. We will refer to this type of behavior as *WO Behavior*, with WO standing for "waypoint list off".

When a WO behavior is performed, the position of the UAV could vary according to direct commands issued by the operator. In this case, the behavior sequence will not describe the path of the UAV until it re-enters autonomous operation mode. Therefore, it would not be

possible to reach the mission's starting point while respecting the restriction that the path to home is known to be safe and collision-free.

In account of this possibility, we define two types of home: *safe home* and *full home*. The operator has authority over the behavior to configure which definition of home should be used by the Fly Home behavior.

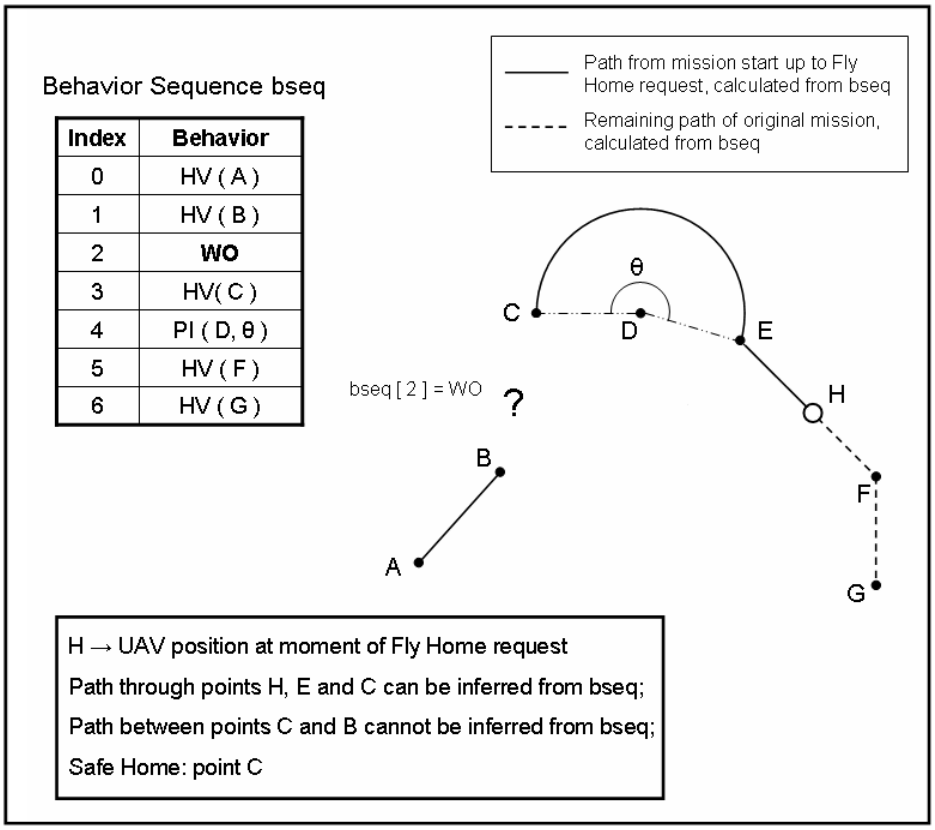

Figure 15: Safe Home

The Safe Home is defined to address the situations in which the operator could use direct position commands to dislocate the UAV during a WO behavior. Therefore, the exact path of the mission cannot be calculated by analyzing the behavior sequence alone. When a Fly Home request is issued, the path between the first waypoint after the WO behavior and the current position can be calculated using the behavior sequence since it is defined by position behaviors. Before this waypoint it is not possible to infer the exact path because of the occurrence of a WO behavior. The Safe Home is then defined by the first waypoint after the last WO behavior before Fly Home was requested. If no waypoint of the mission was reached after the WO task by the moment when Fly Home was requested, Safe Home is the position in such moment. Figure 15 exemplifies the concept of Safe Home.

The formal definition of Safe Home is described in (4-2). This definition is used by the Fly Home behavior to infer the Safe Home position of a mission.

Definition 1: (4-2)

Let *bseq* denote a behavior sequence with *n* behaviors and index $i \in [0, n-1]$

Let $i_{active} \geq 0$ denote the index of the active behavior of *bseq* when the Fly Home request was issued.

Let i_{wo} denote the index such that:

$$i_{wo} = \begin{cases} \text{if } \exists j \in \mathrm{Z} \mid j < i_{active} \wedge bseq_j \text{ is WO} \wedge \nexists\ k \in \mathbb{Z} \mid j < k < i_{active} \text{ and } bseq_k \text{ is WO}\ ,\ i_{wo} = j; \\ \text{else } i_{wo} = 0; \end{cases}$$

Let i_{pos} denote the index such that:

$$i_{pos} = \begin{cases} \text{if } (\exists j \in \mathrm{Z} \mid i_{wo} < \mathrm{j} \leq i_{active} \wedge bseq_j \text{ is a Position Behavior}) \wedge \\ \quad \wedge (\nexists\ k \in \mathrm{Z} \mid i_{wo} < k < j \text{ and } bseq_k \text{ is a Position Behavior })\ ,\ i_{pos} = j; \\ \text{else } i_{pos} = i_{active}; \end{cases}$$

Safe Home: if $i_{pos} < i_{active}$, is the waypoint defined by $bseq_{i_{pos}}$;

else, is the position when Fly Home was requested.

The Full Home is defined to address the situations in which it is assumed that the UAV suffers no dislocation during a WO behavior. This assumption is granted by the operator through the system's configuration. In such circumstances, the path of the mission connecting the position at Fly Home request and the mission's first waypoint can be calculated by analyzing the behavior sequence alone. The Full Home is, hence, defined as the first visited waypoint of the mission. If no waypoint of the mission has been reached at the moment when Fly Home is requested, the current position is the Full Home. The formal definition of Full Home is described in (4-3).

Definition 2: (4-3)

Let *bseq* denote a behavior sequence with *n* behaviors and index $i \in [0, n-1]$

Let i_{active} denote the index of the active behavior of *bseq* when the Fly Home request was issued.

Let i_{pos} denote the index such that:

$$i_{WO} = \begin{cases} \text{if } (\exists j \in \mathbb{Z} \mid 0 \leq j < i_{active} \text{ and } bseq_j \text{ is a Position Behavior}) \wedge \\ \quad \wedge (\nexists k \in \mathbb{Z} \mid 0 < k < j \text{ and } bseq_k \text{ is a Position Behavior}) \;,\; i_{pos} = j; \\ \text{else } i_{pos} = i_{active}; \end{cases}$$

<u>*Full Home*</u>: if $i_{pos} < i_{active}$, is the waypoint defined by $bseq_{i_{pos}}$;

else, is the position when Fly Home was requested.

Figure 16 illustrates practical examples of how to infer the positions of Safe and Full Home using a mission's behavior sequence.

4.3.3 Filtering the Behavior Sequence

On the previous subsections, we described a procedure to analyze which are the waypoints covered by a behavior sequence mission and presented criteria to define where Home is. Now we focus on manipulating the behavior sequence in order to gather information on which tasks were performed by the UAV during the path that connects Home to the position where Fly Home was requested.

The tasks in the behavior sequence which have not been initiated before the Fly Home request are irrelevant to the synthesis of a new mission towards home. The behavior active during the Fly Home request is also irrelevant, unless it is of type Pirouette Around XY. This assertion will be justified later on. The tasks which are Non-Position Behaviors are also irrelevant for the Fly Home behavior, since they do not define waypoints. Another set of behaviors which are irrelevant to the calculation of the mission towards home are the ones denoting tasks which were performed before the UAV reached the Home position. Their irrelevance is justified by the fact that these behaviors define waypoints which should not be

visited when flying home. The pseudo-code of functions 3, 4 and 5 describes the removal of irrelevant behaviors.

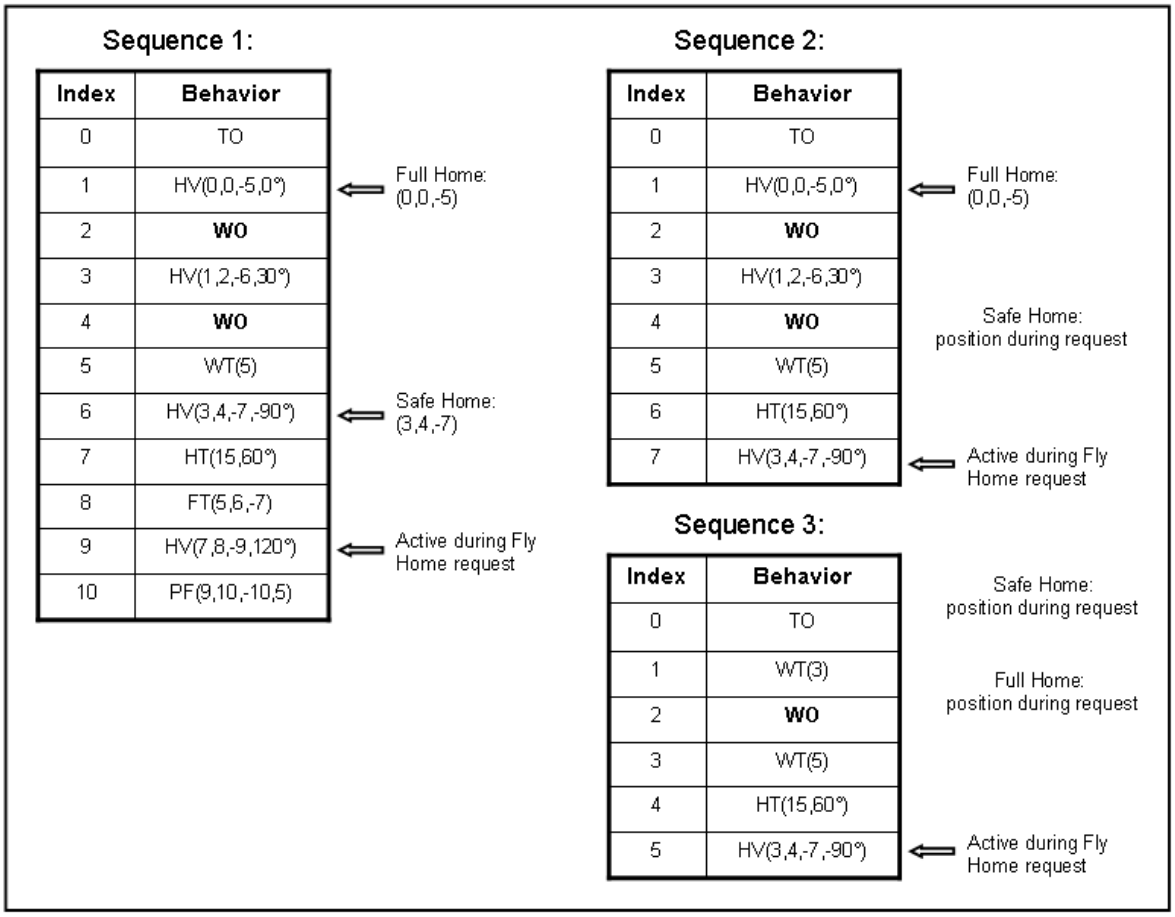

Figure 16: Safe Home and Full Home

```
Function 3:      deleteUnfinished( behaviorSequence )
    Define:
    i_active := index of active behavior during Fly Home request;
    for each behavior behaviorSequence_i
        if i > i_active
            delete behaviorSequence_i
        end if
        if  i = i_active and behaviorSequence_i.type ≠ Pirouette Around XY
            delete behaviorSequence_i
        end if
    end for
```

```
Function 4:    deleteNonPositionBehaviors( behaviorSequence )
    for each behavior behaviorSequence_i
        if behaviorSequence_i is a non-position behavior
            delete behaviorSequence_i
        end if
    end for
```

```
Function 5:    deleteBeforeHome( behaviorSequence )
    Define:
    i_home := index of the behavior which defines the Home position;
    for each behavior behaviorSequence_i
        if i < i_home
            delete behaviorSequence_i
        end if
    end for
```

4.3.4 Inverting the Behavior Sequence

In the previous subsection, we presented procedures to trim the behavior sequence in order to obtain the sequence which position behaviors were performed in order to fly the path from the Home waypoint to the Fly Home request position. The trimmed behavior sequence contains only behaviors which define waypoints. Conformant to the approach proposed for the design of the Fly Home behavior, it is necessary to rearrange the tasks in the trimmed sequence in order to have each waypoint visited in the inverse order.

In order to achieve a behavior sequence with starting point at the Fly Home request position and ending point at the Home position while flying the same path of the original mission, one intuitive possible solution would be inverting the order of the behaviors in the trimmed behavior sequence. The pseudo-code of function 6 describes such an operation.

Although the inverted trimmed sequence is useful to assemble the Fly Home mission, it does not provide a behavior sequence which tasks will lead to a flight towards home following the original path. Several issues have to be addressed for such purpose.

```
Function 6:      invertSequence( behaviorSequence )
     Define:
     auxiliarySequence := behaviorSequence;
     size := behaviorSequence.size;
     behaviorSequence.deleteAllBehaviors();
     for each behavior auxiliarySequence_i
          behaviorSequence_(size-1)-i ← auxiliarySequence_i;
     end for
```

4.3.5 Position Behavior types

With the exception of behaviors of type Pirouette Around XY, in several occasions the other position behaviors are planned offline obeying the restriction that the segment of straight line connecting the starting and ending points of the task must be safe and collision-free. Therefore, it is useful to substitute position behaviors of type Fly To and Pirouette Flight with Hover To behaviors with the same destination coordinates, since Hover To performs the task of hovering from a waypoint to another directly. In case the assumption that the straight line connecting starting and ending points of such position behaviors is not valid, the operator has the possibility of configuring the system to not proceed with the substitute with Hover To behaviors.

```
Function 7:      replaceWithHoverTo( behaviorSequence )
     Define:
     hoverTo := a behavior of Type Hover To;
     for each behavior behaviorSequence_i
          if behaviorSequence_i.type = Fly To or Pirouette Flight
               hoverTo.destination ← behaviorSequence_i.destination;
               behaviorSequence_i ← hoverTo;
          end if
     end for
```

Forth, the term *inverted trimmed behavior sequence* will be used to denote a behavior sequence whose behaviors before the one which defines home, unfinished behaviors and non-position behaviors have been deleted, and the resultant behavior sequence is inverted.

4.3.6 Pirouette Around XY

Some adaptations and corrections are necessary in order to use a behavior Pirouette Around XY in the inverted trimmed sequence to retrace the segment of path described by the pirouette in the original mission. This is mostly due to the fact that the trajectory followed during a behavior of type Pirouette Around XY does not have its starting or ending point specified by the behavior's parameter.

In the following subsections we describe each issue and correction regarding the use of behaviors of type Pirouette Around XY in the inverted trimmed sequence. The last subsection presents a function which addresses all these issues and executes the proper corrections in order to adapt the inverted trimmed sequence for the Fly Home mission.

4.3.6.1 Angular Displacement

As explained in previous sections, the direction of the radial trajectory performed by a Pirouette Around XY task, as well as spline-based Fly-To trajectory, is conformant to the convention that positive angles denote clockwise trajectories and negative angles denote counter-clockwise trajectories. This aspect of the Pirouette Around XY behavior must be remarked when developing a method for computing a Fly Home mission. Consider the situation presented in figure 17.

In the presented situation, if the trajectory is to be followed from point A to C (clockwise) the behavior Pirouette Around XY must have $+\theta$ as angular displacement parameter; if the trajectory must be followed from point C to A (counter-clockwise) the angular displacement parameter must be $-\theta$.

This property implicates that it is possible to retrace backwards a trajectory described by a Pirouette Around XY task. Therefore, all the behaviors of this type in the inverted trimmed behavior sequence from the original mission must have their angular displacement parameter replaced by its additive inverse.

A function which executes such adaptation on the behaviors of type Pirouette Around XY is presented in subsection **4.3.6.4**.

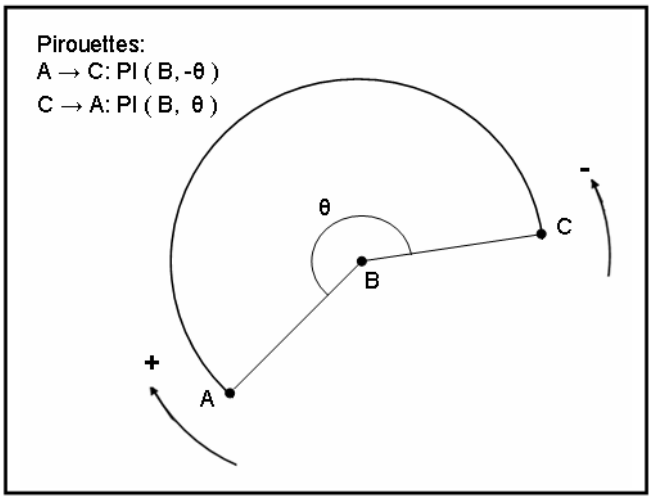

Figure 17: Pirouette direction

4.3.6.2 Start and End Points

Another aspect of the Pirouette Around XY which must be regarded with carefulness is the fact that it does not designate explicitly a waypoint. However, given a starting waypoint, the ending waypoint is designated by the behavior.

Therefore, it is imperative that the starting point of the pirouette is fixed by a previous position behavior. Consider the situations presented in figure 18.

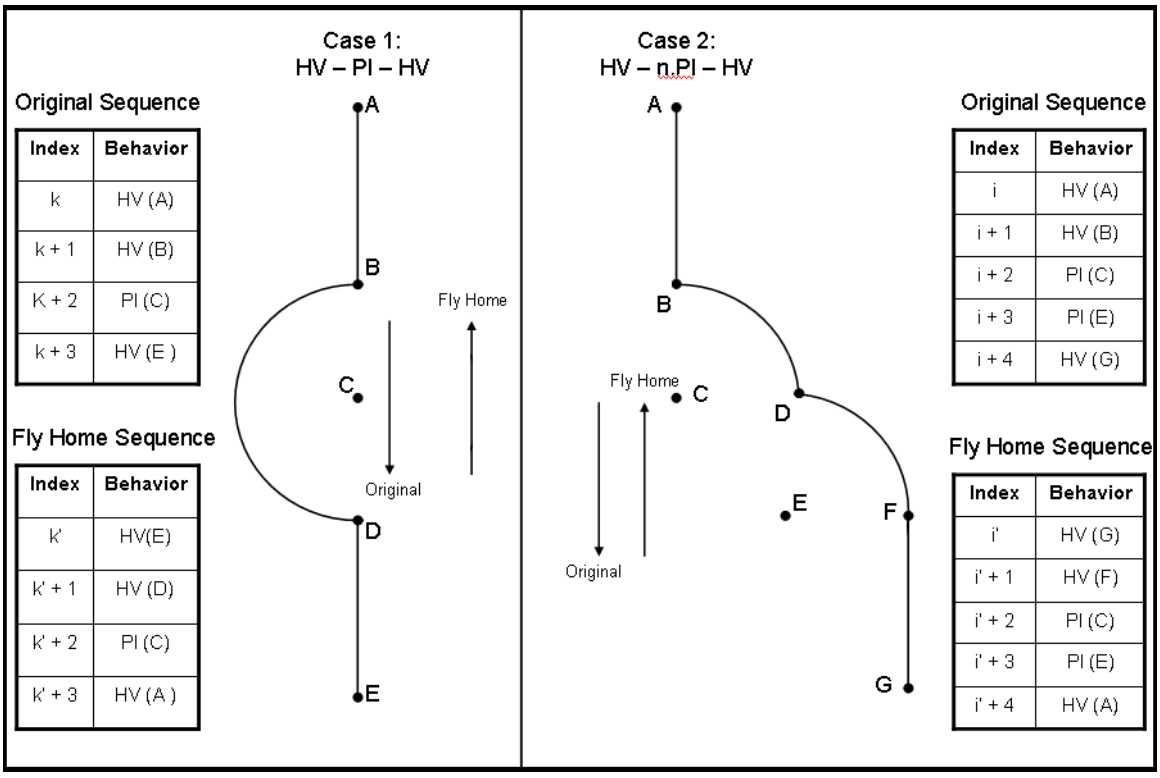

Figure 18: End point of Pirouette Around XY

In Case 1, when inverting the sequence to return from point E to point A, the trajectory observed for the Pirouette Around XY has its ending point on B. Therefore, there is no longer necessity to have a Hover To behavior with destination on point B. However, it is necessary to add a Hover To with destination point D: the point was initially specified as the ending point of the pirouette in the original sequence, and in the inverted sequence must be defined by a Hover To behavior as the starting point of the pirouette.

In Case 2, we observe a similar situation as in Case 1, but with two concatenated behaviors Pirouette Around XY. Once again, once inverting the sequence it is necessary to have the starting point F of the pirouette with center E designated by a behavior Hover To. The starting point of the pirouette with center C is designated by the former pirouette with center E. The waypoint B is already specified by the pirouette with center C, so it is no longer necessary to have a behavior Hover To point B. The example demonstrates that when the inverted trimmed sequence comprises a set of concatenated Pirouette Around XY tasks, it is necessary to add a Hover To task to designate the starting point of the first Pirouette task of the set. Moreover, the Hover To task designating the waypoint at the end point of the last Pirouette of the set is redundant and can be removed.

```
Function 8:     adaptPiEndPoints( behaviorSequence )
    Define:
    hoverTo := a behavior of Type Hover To;
    for each behavior behaviorSequence_i of type Pirouette A. XY
        if behaviorSequence_i-1 .type ≠ Pirouette Around XY
                hoverTo.destination ← start point of current
                                      pirouette;
            insert hoverTo between behaviorSequence_i and
                                      behaviorSequence_i-1;
        end if
        if behaviorSequence_i+1 .type ≠ Pirouette Around XY
            delete behaviorSequence_i+1;
        end if
    end for
```

The pseudo-code of function 8 describes the insertion, when necessary, of Hover To tasks to designate the starting point of Pirouette Around XY tasks and to delete, when possible, Hover To tasks designating the same waypoint as Pirouette Around XY tasks.

4.3.6.3 Fly Home Request during Pirouette Around XY

If a Fly Home mission request is issued during any task except Pirouette Around XY, hovering to the last waypoint visited is already a valid solution, since the straight line connecting this point and the next waypoint is clear of obstacles. However, this assumption is not true when Fly Home is requested during a Pirouette Around XY. Therefore, it is necessary to retrace the pirouette starting from the point where Fly Home was issued.

Since in this case the pirouette task was not accomplished, the angular displacement during the trajectory was smaller in module than the one described by the behavior's parameters. Hence, it is necessary to calculate the value of the angular displacement until the UAV comes to rest and adapt the pirouette's parameters. Figure 19 exemplifies the idea.

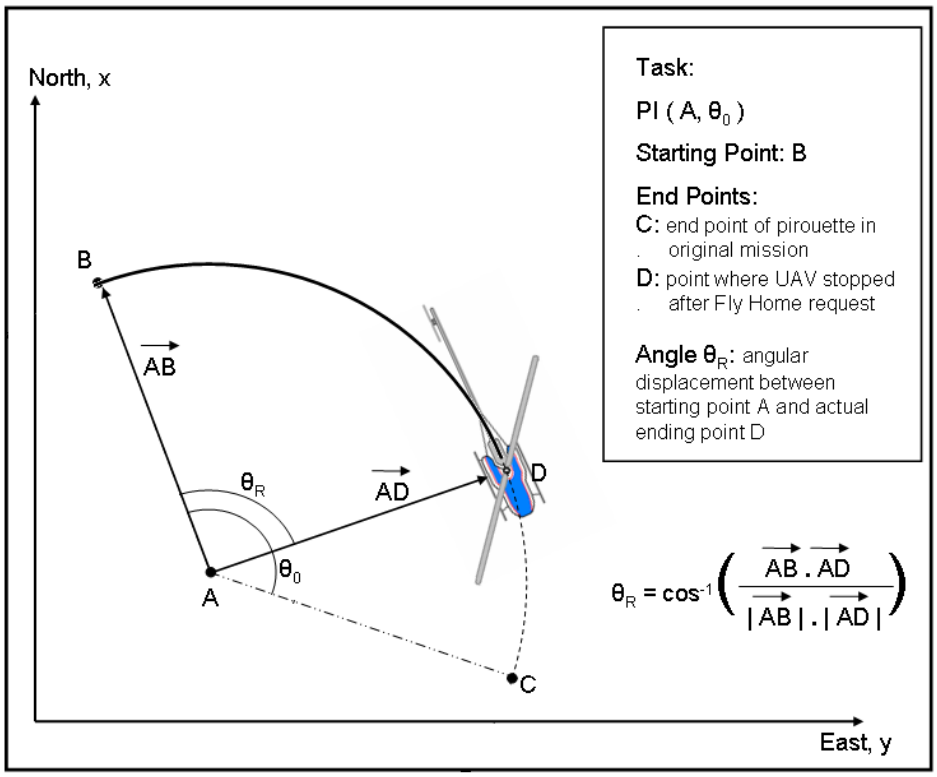

Figure 19: Angular displacement in an interrupted Pirouette Around XY task

Forth, the problem of flying back to the waypoint which preceded the interrupted Pirouette Around XY task in the original mission following the original trajectory is

addressed. The solution proposed is adapting the considered pirouette behavior in the inverted trimmed behavior sequence in an analogous way as the one described by subsection **4.3.6.1**. The difference consists in substituting the behavior's angular displacement parameter with the additive inverse of the actual angular displacement calculated as shown in figure 19.

The pseudo-code of function 9 was designed to adapt the angular displacement parameter of all the behaviors of type Pirouette Around XY in the inverted trimmed sequence. This function addresses the issues presented on subsections **4.3.6.1** and **4.3.6.4**.

4.3.7 Robustness and Performance Efficiency

The subsections presented above list issues concerning the re-planning of the original mission in order to fly home through the original trajectory. The functions mentioned in these subsections are enough to assure that such an objective is successfully achieved.

However, some aspects of the mission leave room for improvement with respect to robustness and performance efficiency. The following subsections address these issues and present improvements.

```
Function 9:     adaptPiAngularDisplacement( behaviorSequence )
    Define:
    float adaptedAngularDisplacement;
    i_active := index of active behavior during Fly Home request;
    for each behavior behaviorSequence_i of type Pirouette A. XY
        if i = i_active
            adaptedAngularDisplacement ← - actual angular displacement;
        end if
        else
            adaptedAngularDisplacement ←
             - behaviorSequence_i .angularDisplacement;
        end else
        behaviorSequence_i .angularDisplacement ← adaptedAngularDisplacement
    end for
```

4.3.7.1 Heading: Hover To Behavior

As discussed in subsection **4.3.5**, substituting the position behaviors in the inverted sequence with Hover To behaviors designating the same waypoint is beneficial to the performance of the Fly Home mission. Another aspect to be considered is the heading associated with these behaviors.

The performance efficiency of the UAV's actuators is optimal when its velocity vector and heading are aligned, since the fuselages of ARTIS' helicopters have been designed to reduce drag when flying facing forwards. Hence, such heading would also improve overall performance efficiency of the Fly Home mission.

Moreover, during the flight, exists the possibility that ARTIS' Visual Collision Avoidance system is in operation [25]. In such circumstances, maintaining the heading aligned with the UAV's velocity vector would allow the collision avoidance system to detect and prevent a collision with an eventual obstacle in the path which was not previously regarded when planning the original mission. Therefore, such heading would improve overall safety of the Fly Home mission.

Considering that the heading aligned with the velocity vector enhances safety and performance of operation, this heading was the choice for the Hover To behaviors on the Fly Home mission. To achieve such heading, it suffices to set the behavior's heading parameter parallel to the segment of straight line connecting the starting and ending waypoints of the Hover To task. These points can be retrieved in the waypoint array generated by the function *computeWaypoints(.)*, presented on subsection **4.3.1**, applied to the inverted trimmed sequence.

```
Function 10:    setHvHeading( behaviorSequence, waypoints )
    Define:
    float heading;
    for each behavior behaviorSequence_i of type Hover To
        heading ← inclination of position vector (waypoint_i –
                waypoints_i-1;
        behaviorSequence_i .heading ← heading;
    end for
```

4.3.7.2 Heading: Pirouette Around XY Behavior

The behavior type Pirouette Around XY was originally designed to allow the UAV to hover around an object in the center of the trajectory while keeping its heading pointed at the object in order to take pictures or film it.

However, when flying Home, this application no longer applies. Therefore, it is necessary to establish criteria to define the heading for the tasks of type Pirouette Around XY.

The objective of the intelligent behavior Fly Home is to return to Home safely. Therefore, the chosen criteria according to which the heading should be defined are the enhancement of the safety and of the performance efficiency of the Fly Home mission.

As explained on subsection **4.3.6.1**, the choice of heading tangent to the trajectory of the UAV enhances the overall performance and safety of the mission. Considering these arguments, the heading chosen for behaviors of type Pirouette Around XY in the Fly Home mission is aligned with the velocity vector of the UAV. Figure 20 presents the different desired headings according to the purpose of the mission.

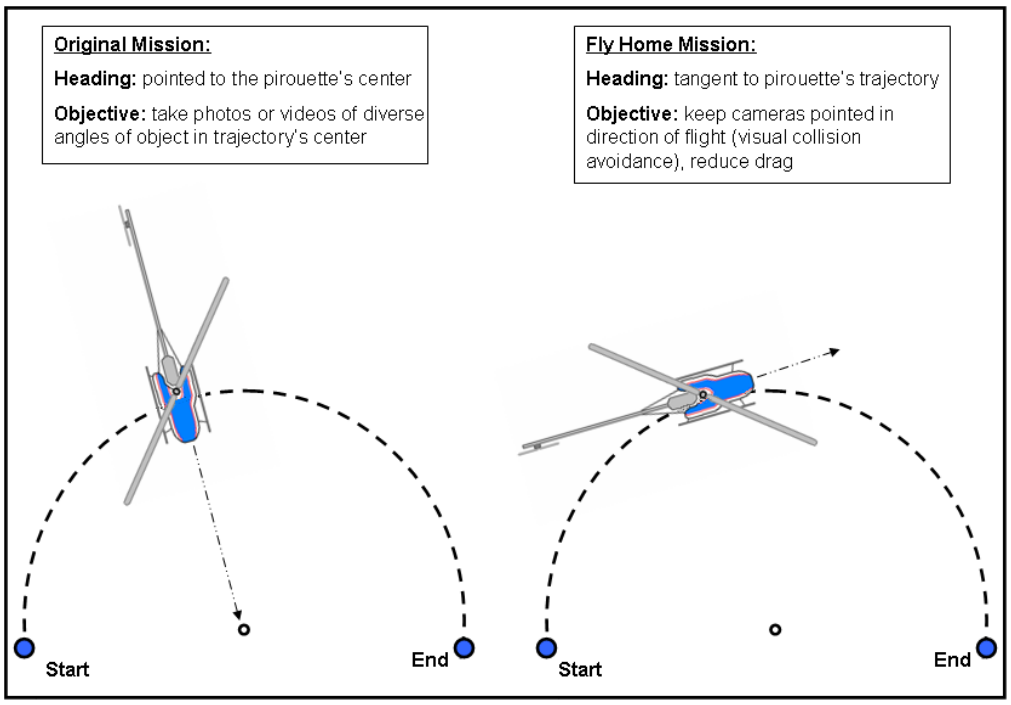

Figure 20: Pirouette Around XY heading possibilities

The Pirouette Around XY maintains the angle between the heading vector and the radial direction of the trajectory. Hence, in order to set the right heading on the starting point of the pirouette, it is sufficient to have before it a Hover To task with destination to the

starting point and heading tangent to the circular trajectory. Figure 21 presents the calculation of such heading.

A function regarding the issue of introducing a Hover To behavior which is in charge of setting the right heading before the beginning of the following Pirouette Around XY task is presented in subsection **4.3.7.4**.

4.3.7.3 Position Behaviors Starting Point

As presented in previous sections, the trajectory of a position behavior task depends both on the starting point of the mission and the parameters of such behavior. The path described by a Pirouette Around XY task is specially sensitive to variations on its starting point. The radius of the radial trajectory is calculated based on the distance between the Pirouette's center point (described by its parameters) and the starting point of the task. Therefore, deviations in the intended starting point result in differences between the arc trajectory (including starting and ending points) intended by the operator and the one traced online.

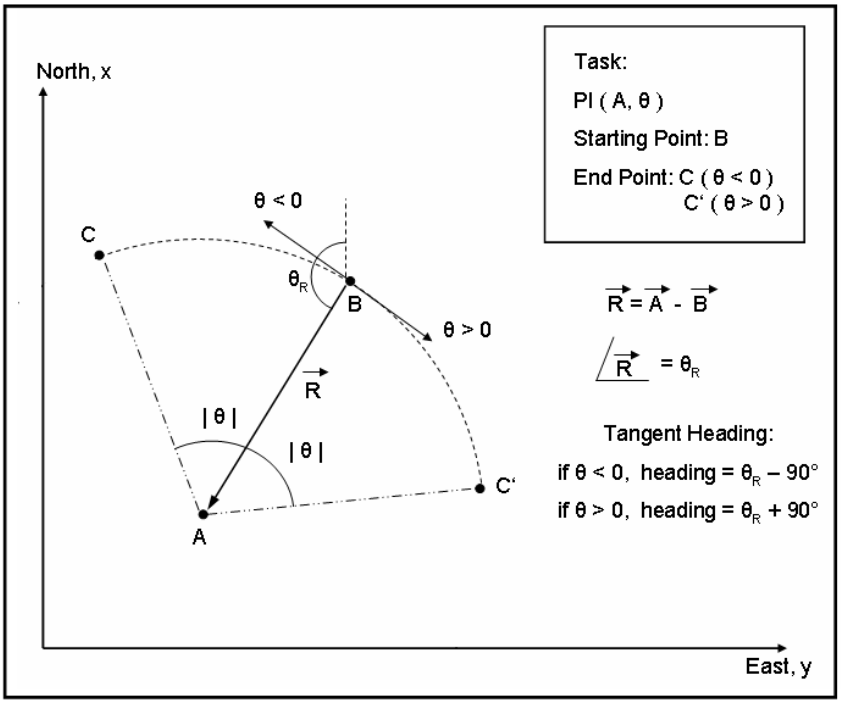

Figure 21: Heading tangent to pirouette trajectory

Reducing deviations between the Fly Home path and the intended original path connecting Home and the point where Fly Home was requested increases the overall safety of

the mission. Therefore, it would be beneficial to the Fly Home mission to increase the robustness with respect to path deviations caused by unintended tasks' starting points.

The discussed robustness is enhanced when, at the moment of the beginning of a new task, the UAV hovers from its current position to the intended starting point of the respective position behavior. Such solution can be achieved by performing a Hover To task with destination to the intended starting point of the behavior. A function addressing this issue is presented on subsection **4.3.7.4**.

4.3.7.4 Heading Setting Before Task

When starting a Hover To task, the deviation between the intended path and the actual path flown is greater if the heading of the UAV at the moment when the task begins is different from the heading specified by the behavior's parameter. Figure 22 exemplifies such discrepancies.

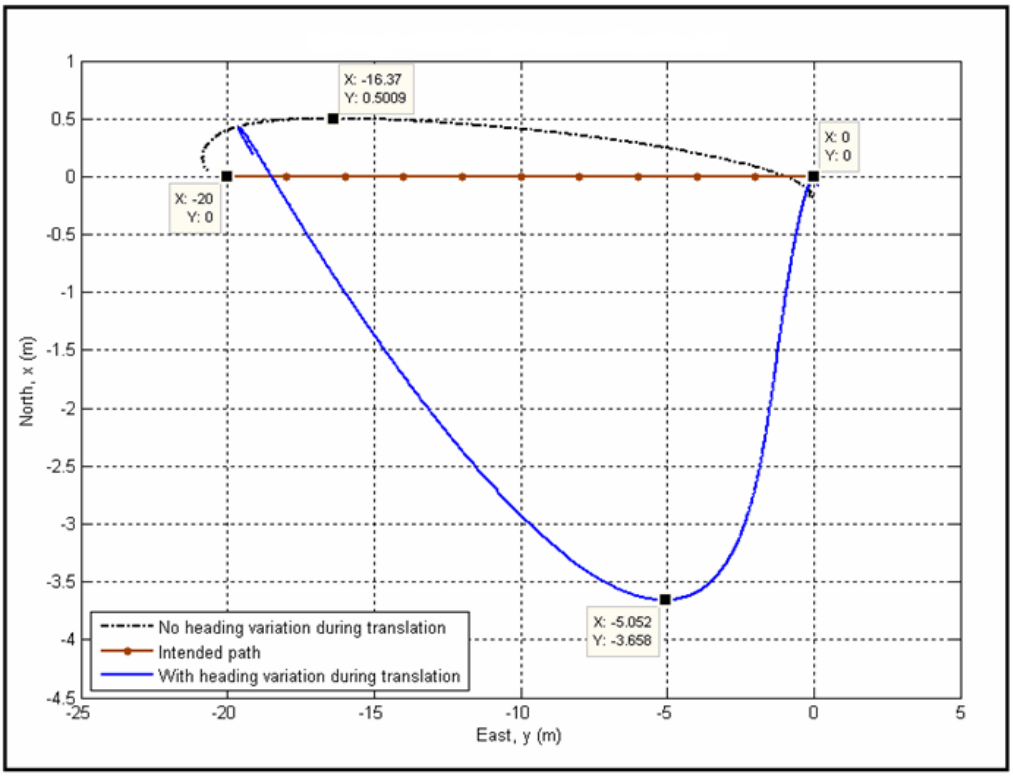

Figure 22: Super-positioning heading changes while moving

Figure 22 depicts the trajectory in x and y direction of two missions executed using Software-in-the-Loop simulation. On both cases the mission consists in hovering from waypoint (0, 0, -5) to (0, -20, -5) with heading 180°. In one of them the initial heading is 0°

and the heading was adjusted during the Hover To task. In the other case the initial heading was already 180°, so the system actuated simply to maintain heading. In the figure the points of maximum deviation in the x direction from the intended trajectory is marked, as well as the intended starting and ending points.

As shown in the figure, the maximum deviation in the x direction was 7.30 times greater in the case which the heading started at 0°. This example demonstrates that deviations between intended and actual trajectories can be reduced if the heading before a Hover To task is the same as the heading defined by the behavior's parameters. The system's robustness against deviation caused by heading changes would hence increase.

```
Function 11:    setRobustnessHvTask( behaviorSequence )
    Define:
    hoverTo := a behavior of Type Hover To;
    for each behavior behaviorSequence_i of type Pirouette A. XY
        hoverTo.destination ← startingPoint of behaviorSequence_i;
        hoverTo.heading ← tangent to trajectory of
                          behaviorSequence_i;
        insert hoverTo between behaviorSequence_i and behaviorSequence_i
    end for
    for each behavior behaviorSequence_i of type Hover To
        hoverTo.destination ← startingPoint of behaviorSequence_i;
        hoverTo.heading ← behaviorSequence_i .heading;
        insert hoverTo between behaviorSequence_i and behaviorSequence_i-1
    end for
```

Setting the heading of the UAV to the one described by a following Hover To behavior's parameters can be achieved by inserting before this behavior another Hover To task with destination to the intended starting point and heading set as mentioned before. This solution also applies to the issue described in subsection **4.3.7.3**, since the robustness Hover To task's destination is the position behavior's intended starting point. If the same solution is applied to Pirouette Around XY behaviors with the heading tangent to their trajectories, the issue described in subsection **4.3.7.2** is also contemplated.

The pseudo-code for function 11 was designed to address the problems described in subsections **4.3.7.2**, **4.3.7.3**, **4.3.7.4**.

4.3.7.5 Proximity between Waypoints

In a regular mission planned by GCS, there is the possibility that two or more consecutive position behaviors designate waypoints within a proximity considered safe. The behaviors in a mission whose designated waypoints are in such situation will be referred to as *redundant set of behaviors*. This is common in situations when the mission planner wants the helicopter to reach a specific absolute heading without translation. The threshold within which the distance between waypoints is considered to be safe is consonant to the standards used by the embedded system.

In the scenario described above, it is ineffective in the Fly Home mission to have redundant sets of behaviors. This aspect would translate into more time being spent with tasks which will not result into improvement of either overall safety or performance efficiency of the Fly Home mission. Therefore, eliminating redundant sets of behaviors would be beneficial to the efficiency of the mission. In order to achieve such purpose, one solution would be to selectively remove behaviors from the inverted trimmed behavior sequence that tasks so that each waypoint is defined by only one position behavior, and the distance between consecutive waypoints is above the threshold of safety.

```
Function 12:    removeWaypointRedundancies( behaviorSequence, waypoints )
    for each behavior behaviorSequence_i
        if   behaviorSequence_i .destination is too close to
             behaviorSequence_i-1 .destination
             delete behaviorSequence_i;
             delete waypoints_i;
        end if
    end for
```

Function 12 was designed to achieve the aforementioned purposes. In the scope of the computation of the Fly Home mission, it is important to point out that this function should be

used before adding the robustness Hover To tasks (previous subsection); otherwise the newly added robustness behaviors are deleted.

4.3.8 Other Issues

This subsection is intended to provide solutions regarding remaining minor issues which must be addressed before finishing the Fly Home mission planning.

The first issue is the case in which helicopter is landed at the moment of the Fly Home request. In this situation, the mission towards home can be calculated normally, and after the behavior sequence has been computed, a Take Off behavior is inserted in the sequence as its first task.

```
Function 13:    concludeSequence( behaviorSequence )
    if behaviorSequence.size = 0
        Define slowDown := a behavior of type Slow Down;
        behaviorSequence0 ← slowDown;
        alert system that it's  impossible to plan Fly Home
        mission;
    else if UAV was landed during Fly Home request
        Define takeOff := a behavior of type Take Off;
        insert takeoff before first behavior of behaviorSequence;
    end if
    if UAV was at home during Fly Home request
        empty behaviorSequence;
        alert system that home has already been reached;
    end if
```

Another issue would be the case in which the planning of the mission towards home generated an empty behavior sequence. This problem occurs in missions in which no position behavior could be completed up until the Fly Home request, or missions which start with a Pirouette Around XY as their first position behavior, since the pirouette's starting and ending point would, in such cases, be indefinable through the behavior sequence alone. If the Fly Home mission planning resulted in an empty behavior sequence, we add a single task of type

Slow Down to the sequence (for safety, in case the UAV is moving) and alert the system that no valid Fly Home mission could be calculated.

A third issue to be addressed is the situation in which the UAV's position at the moment of the Fly Home request is already within a safe proximity of the location designated as Home. In this particular case, the objective of the Fly Home mission is completed and there is not necessity to execute further tasks. Under these circumstances, no task is defined by the Fly Home behavior and the system is alerted that the UAV has reached home.

The pseudo-code for function 13 addresses the issues discussed in this subsection.

4.3.9 Fly Home Mission Algorithm

The definitions and functions described in previous subsections make it possible to assemble an algorithm designed to calculate a behavior sequence describing a Fly Home mission from the behavior sequence representing the respective mission.

The proposed algorithm requires no path-planning, world model or additional history-keeping modules. The re-planning towards home can be performed online based on information provided by a mission's behavior sequence. The path performed during the execution of the tasks of the Fly Home mission does not cross areas whose safety is unknown. Therefore, all the requirements presented on section **4.1** have been successfully fulfilled. The algorithm proposed has time complexity of order O(n).

4.4 Managing the Fly Home Behavior

The former section presented a method to compute a Fly Home mission based on the original mission performed. However, planning the flight back to home is not sufficient to achieve the goals of the behavior. In order to execute properly the Fly Home behavior, it is imperative to provide the necessary means for the UAV to respond to a Fly Home request, to re-plan the mission, to execute its mission in a safe and effective manner and to handle possible failures or external requests properly.

This section presents the module responsible for managing the different stages of a Fly Home behavior. The management takes account of safety and performance efficiency of the behavior.

```
Algorithm: Fly Home Mission Computation
      Define:
      originalSequence: The behavior sequence of the original
                        mission
      homeSequence: The behavior sequence of the Fly Home mission
      waypoint: Array of waypoints described by inverted trimmed
            behavior sequence
      i_home: index of behavior which designates Home waypoint of
            originalSequence
      i_active: index of originalSequence

Pseudo-code: Fly Home Mission Computation
      homeSequence ← originalSequence;
      deleteBeforeHome( homeSequence );
      deleteNonPositionBehaviors( homeSequence );
      deleteUnfinished( homeSequence );
      computeWaypoints( homeSequence );
      invertSequence( homeSequence );
      invert waypoint array;
      removeWaypointRedundancies( behaviorSequence, waypoints );
      replaceWithHoverTo( behaviorSequence );
      adaptPiEndPoints( behaviorSequence );
      adaptPiAngularDisplacement( behaviorSequence );
      setHvHeading( behaviorSequence, waypoints );
      setRobustnessHvTask( behaviorSequence );
      concludeSequence( behaviorSequence );
```

Forth, the following terms will be used to denote different aspects of the Fly Home behavior:

- *Fly Home Mission*: the sequence of tasks defined by the behavior sequence generated by the Fly Home behavior.
- *Fly Home State*: the module responsible for managing the events which could take place during a Fly Home behavior and the different stages on the temporal evolution of the behavior.

- *Fly Home Object*: the module responsible for realizing the functionalities required by the Fly Home behavior. This denomination is inspired by the implementation of the behavior using Object Oriented Programming.

4.4.1 Fly Home Behavior Modeling

The Fly Home behavior was modeled according to UML specifications for Statechart diagrams. This choice was based on the advantages of statechart-based modeling as presented on subsection **3.1.1**. This model was conceived with the purpose of managing possible events during a Fly Home mission, as well as allowing the UAV to execute different stages of the behavior. As stated in section **3.1**, the Fly Home state is part of the statechart diagram designed to model the Supervisor module. Figure 23 presents the statechart model designed for the Fly Home behavior.

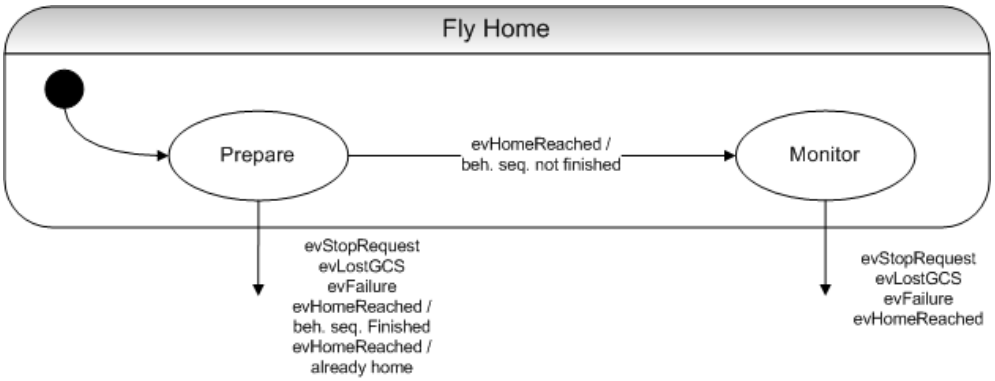

Figure 23: Fly Home statechart model

The Fly Home state comprises two sub-states, namely *Prepare* and *Monitor*. The default sub-state of Fly Home state is Prepare sub-state. Bellow follows a brief description of each sub-state.

- *Prepare*: the system transits to this sub-state at the beginning of the Fly Home behavior. When entering the sub-state, the Fly Home object uses the behavior sequence describing the original mission to compute the Fly Home mission. While the Fly Home mission is executed by the system, it remains in Prepare sub-state.

- *Monitor*: the system transits to this sub-state when, while in sub-state Prepare, the UAV reaches home before expected. In the state chart above, this event is denoted by *evHomeReached / beh. seq. not finished* and is more thoroughly explained ahead in this

subsection. In the sub-state Monitor a new mission is computed destined to stop the UAV and direct it straight to Home location.

The two sub-states presented are described in more detail in subsections **4.4.1.1** and **4.4.1.2**.

Below follows the description of the events relevant during while the system is in Fly Home state.

- *evStopRequest*: occurs when GCS issues a Stop command to the UAV;
- *evLostGCS*: occurs when the UAV's embedded system looses contact with GCS and the operator;
- *evFailure*: occurs when the Fly Home object is not able to calculate the mission towards home, e.g. when no position behavior was executed after a WO behavior or when the first position behavior of the original mission is of type Pirouette Around XY, for neither starting or ending points can be inferred from the behavior sequence.
- *evHomeReached*: occurs when the distance between the UAV's position and Home position is within a pre-defined threshold. Three different guard conditions for this event exist:
    1. *beh. seq. not finished*: behavior sequence not finished i.e. the Fly Home mission has not been performed until completion when the UAV has reached Home position.
    2. *beh. seq. finished*: behavior sequence finished i.e. the Fly Home mission has been performed until the last of its tasks has been completed successfully.
    3. *already home*: the UAV is already at Home when Fly Home is requested.

4.4.1.1 Prepare

The sub-state Prepare is in charge of computing and managing the Fly Home mission, which is calculated upon the sub-state's entry using the algorithm presented on subsection **4.3.7**.

Some functionalities of the behavior are executed by the Fly Home object in this sub-state. A list containing functionalities addressed within the Prepare sub-state scope is presented below.

- Original Mission: one feature of the Fly Home behavior is to make possible for the UAV to re-start the original mission starting from the behavior which designates Home after home has been reached. Therefore, during this sub-state the Fly Home object saves the original mission when entering. When Home is reached, it resets the original behavior sequence with internal index pointing at the behavior which designates Home.
- Aborting mission: if one of the events evStopRequest, evLostGCS or evFailure (all described in **4.4.1**) occurs, the Prepare sub-state is in charge of aborting the mission and alerting the operator about the cause of abortion.
- Reaching Home: if the event evHomeReached occurs, the sub-state is responsible of recognizing and informing the operator about the conditions of such event.

The flowchart presented in figure 24 illustrates the processes and features under responsibility of Prepare sub-state.

4.4.1.2 Monitor

The Monitor sub-state is entered when the UAV reaches the specified Home location before the Fly Home mission is fully accomplished. In other words, Monitor is activated when the UAV achieves its objective before expected.

For the sake of performance efficiency, under these circumstances it is undesirable that the UAV finished the computed Fly Home mission, since the vehicle will visit several waypoints before returning to Home. Therefore, the objective of this sub-state is to manage both the abortion of the regular Fly Home mission and the start of a new leading mission directly to Home.

The features of the Monitor sub-state comprise the items *Original Mission* and *Aborting Mission* described in subsection **4.4.1.1**. The flowchart presented on figure 25 illustrates the processes and features under responsibility of Monitor sub-state.

The *Back to Home* mission present on the Monitor flowchart is the behavior sequence responsible of commanding the UAV to stop, turn around and go back to the Home position. Figure 26 illustrates the Back to Home mission.

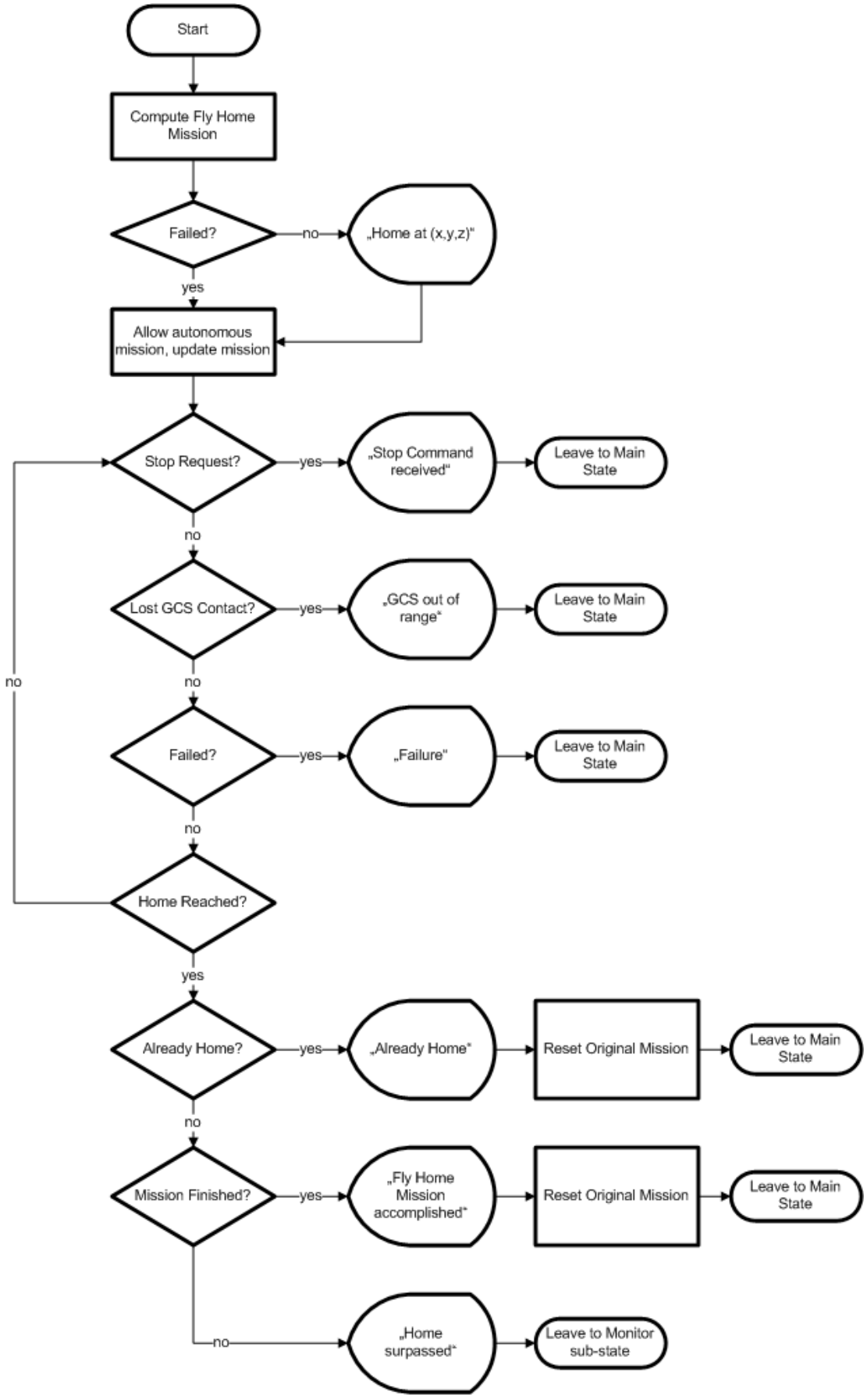

Figure 24: Prepare sub-state flowchart

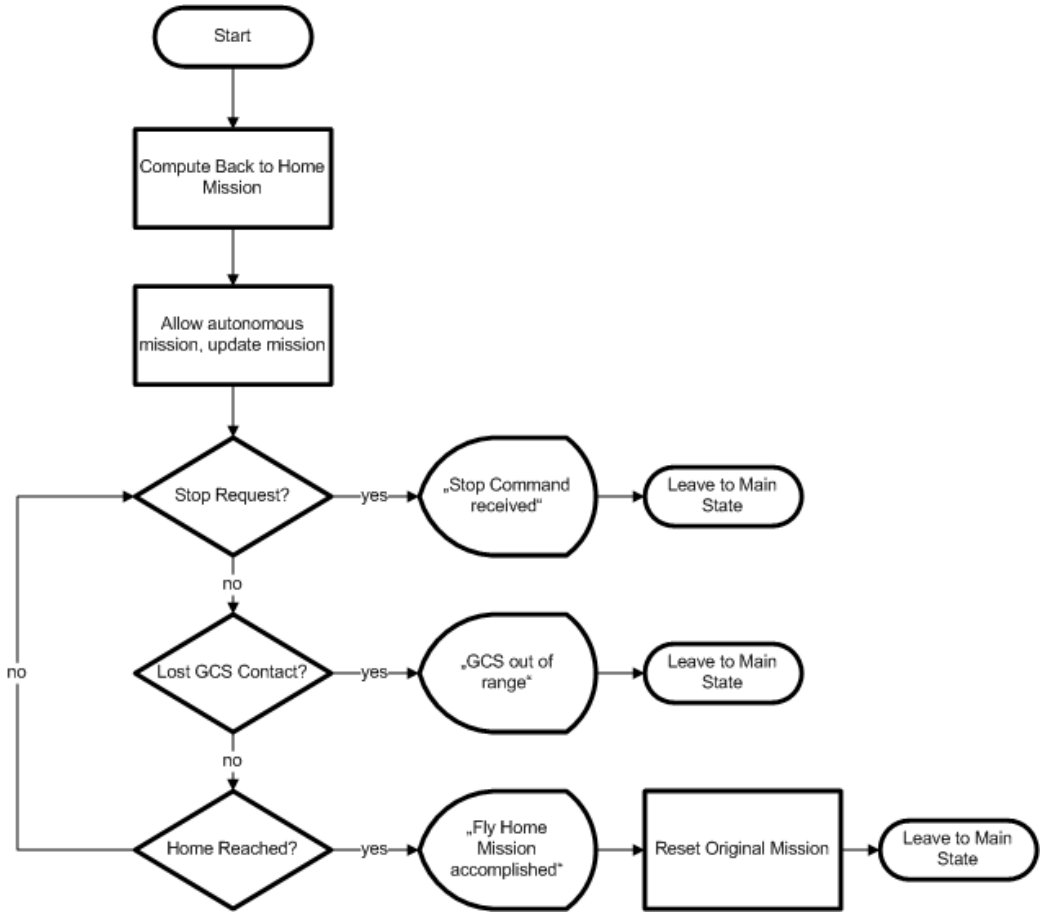

Figure 25: Monitor sub-state flowchart

The left part of figure 26 describes the situation which Monitor state is designed for. The intended path of the Fly Home mission crosses the limits of a ball $B_r(H)$ centered at Home waypoint H with radius r defined by the system's threshold within which a position is considered to be reached. After the UAV reached the ball $B_r(H)$ when executing the regular Fly Home mission, the Supervisor transits to Monitor sub-state, which calculates and sets the Back to Home mission. The first behavior of the mission is intended to slow down the UAV until stillness; the second is responsible to turn the UAV's heading around at the point it stopped; the third commands the UAV to hover from the stop point to Home with heading pointed towards Home.

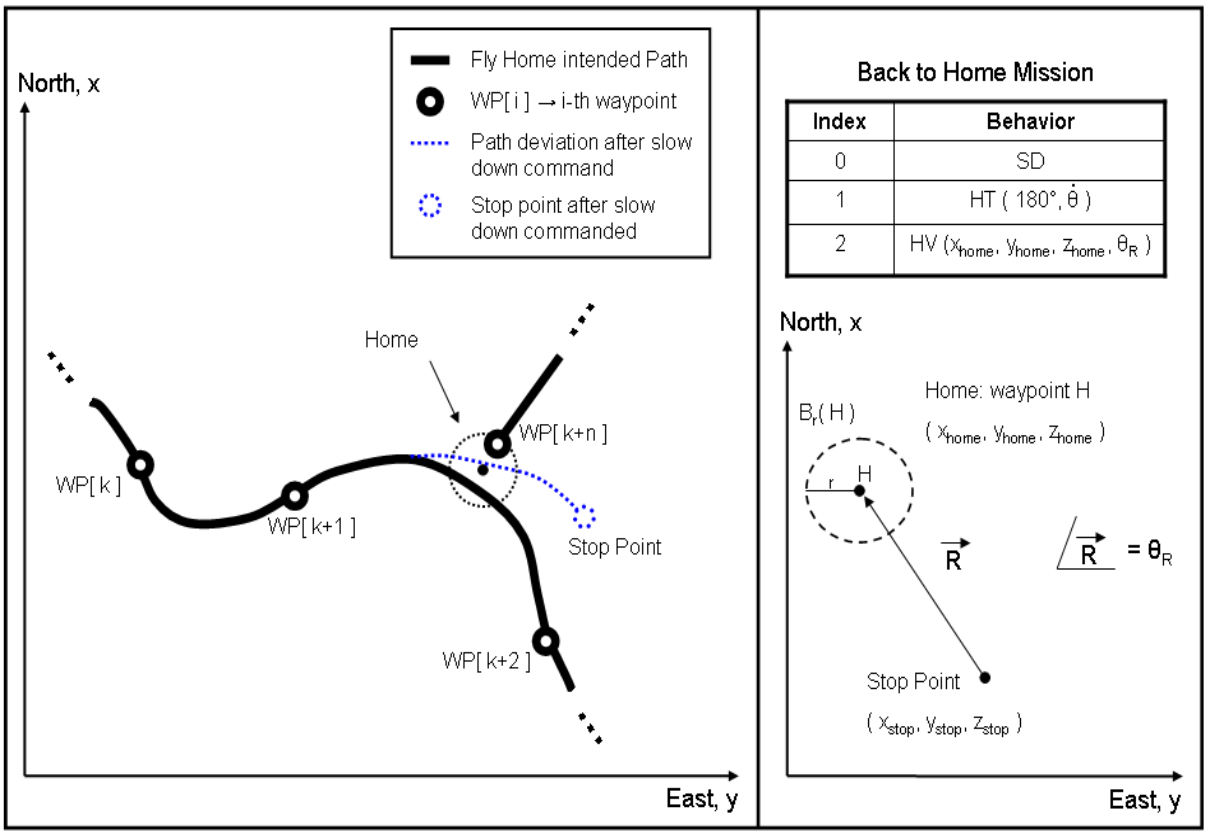

Figure 26: Monitor situation, Back to Home mission

4.5 Results

This section presents the practical results obtained with the implementation of the Fly Home behavior, as well as the analysis of such outcome. It is important to remember that, unless explicitly remarked otherwise, the coordinates shown in this section are conformant to the NED reference system, i.e. (North [m], East [m], Down [m]) set at ground level. Therefore, heights will be negative, e.g. the coordinates (5, 3, -2) [m] represents 5 meters north to the reference, 3 meters east and 2 meters of altitude with respect to ground level.

4.5.1 Fly Home Mission Planning

To test the functionalities of the Fly Home behavior mission planning capabilities, an original mission was planned offline and three different scenarios were tested. In each scenario, a Fly Home request is issued in a different stage of the mission under different Home configurations, i.e. *Safe Home* and *Full Home*. The validity of the results was assured using Unit Testing [26].

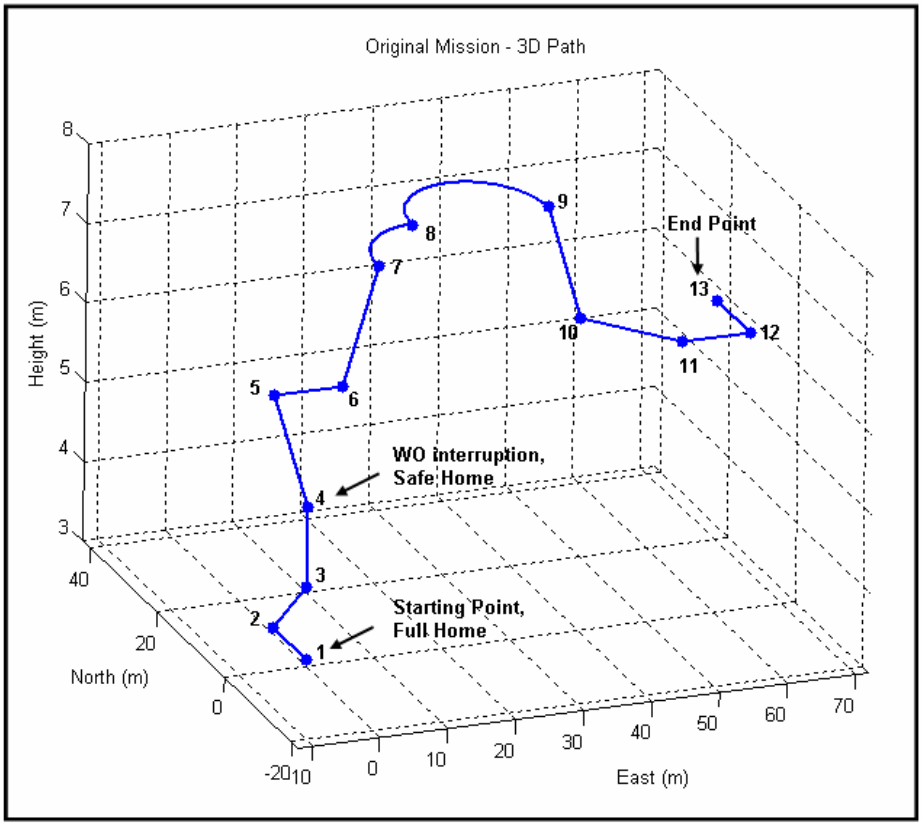

Figure 27: Original mission's path, 3D representation

The mission planned is described in figure 27. The figure presents the graphic tri-dimensional representation of the path described by the original mission. The graph uses positive height (i.e. non-NED) in order to avoid misunderstandings. The waypoint list representing the mission is available at the appendix.

In order to facilitate the visualization of the mission, we also present the bi-dimensional representation of the mission's path in figure 28.

As shown, the mission comprises 13 waypoints. The starting waypoint of the mission (and therefore, its Full Home) is waypoint 1 and the ending waypoint is waypoint 13. After reaching waypoint 3, the behavior sequence activates a WO behavior before hovering to waypoint 4. Therefore, when Fly Home is requested *after* the system has achieved waypoint 4, the designated Safe Home of the Fly Home mission is waypoint 4.

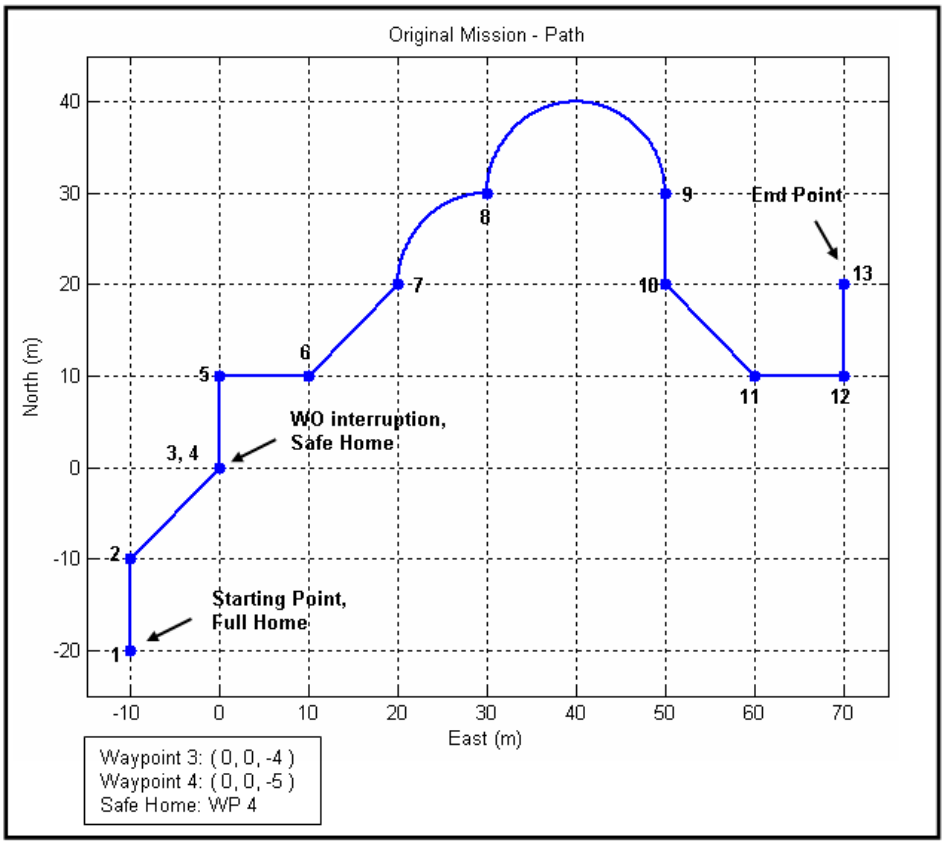

Figure 28: Original mission's path, 2D representation

The mission includes two behaviors of type Pirouette Around XY and eleven position behaviors of other types. Also, the behavior sequence contains several non-position behaviors throughout the mission.

4.5.1.1 First Scenario

In the first scenario, the mission is executed from its starting behavior until the Hover To behavior leading from waypoint 11 to waypoint 12. During the execution of such behavior, a Fly Home request is received and the mission is interrupted. In such moment, the UAV's position is at point A = (10, 65, -6) [m]. The configuration of the system is set to restrict the Home position with Safe Home definition.

In such scenario, we applied the mission planning capabilities of the Fly Home behavior to plan a mission towards home. The path of the resultant mission is shown in figure 29.

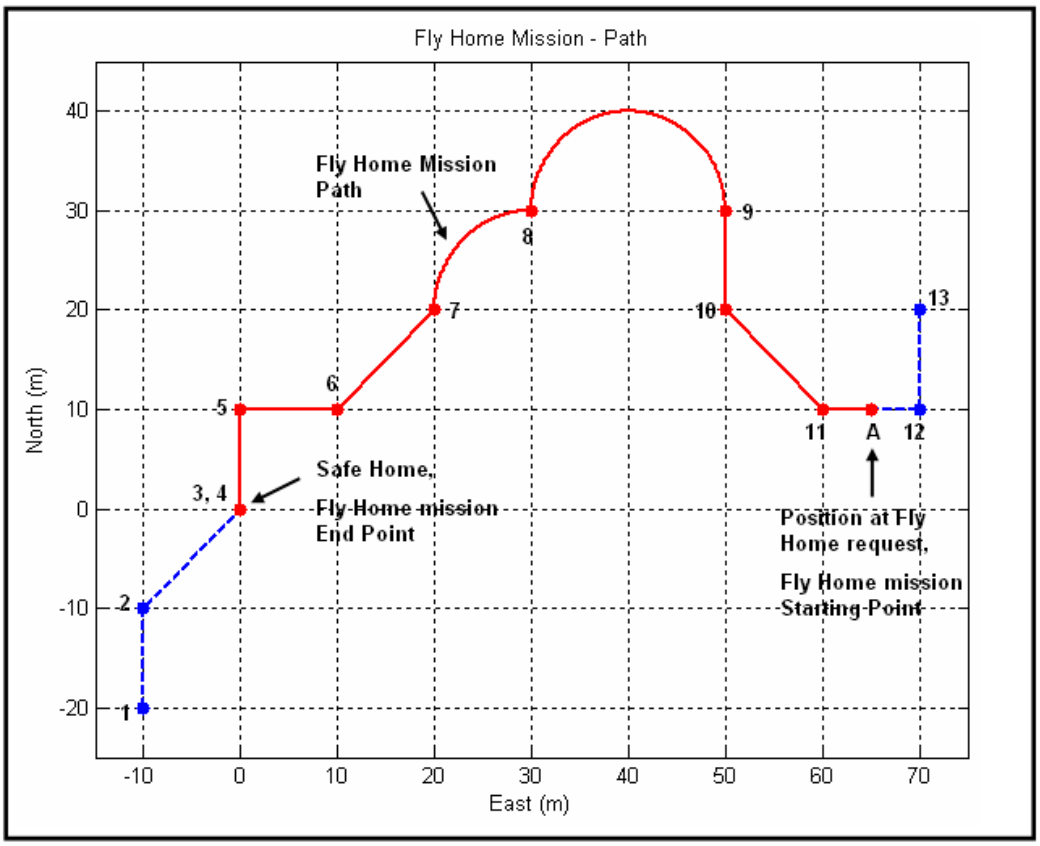

Figure 29: Fly Home mission's path, scenario 1

As illustrated in figure 29, the Fly Home mission starts on point A and ends on waypoint 4, the Safe Home of the mission. Furthermore, the path of the mission is such that it visits waypoint 11 till 4 sequentially and the path's points are conformant with the original mission's path. Moreover, other aspects of the Fly Home mission not shown in the picture (e.g. waypoints' height, position behaviors' headings, behaviors in charge of setting the right heading before translations, starting points of pirouettes and initial heading properly designated, filtering of non-position behaviors) have all been validated using Unit Testing. The waypoints before Safe Home position (waypoint 4) and after point A have not been visited by the Fly Home mission's path. The waypoint list of the mission generated by the Fly Home behavior can be found in the appendix.

4.5.1.2 Second Scenario

In the second scenario, the mission is executed from its starting behavior until the Pirouette Around XY behavior leading from waypoint 8 to waypoint 9. During the execution of such behavior, a Fly Home request is received and the mission is interrupted. In such

moment, the UAV's position is at point B = (40, 40, -7) [m]. The configuration of the system is set to restrict the Home position with Safe Home definition.

In such scenario, we applied the mission planning capabilities of the Fly Home behavior to plan a mission towards home. The path of the resultant mission is shown in figure 30.

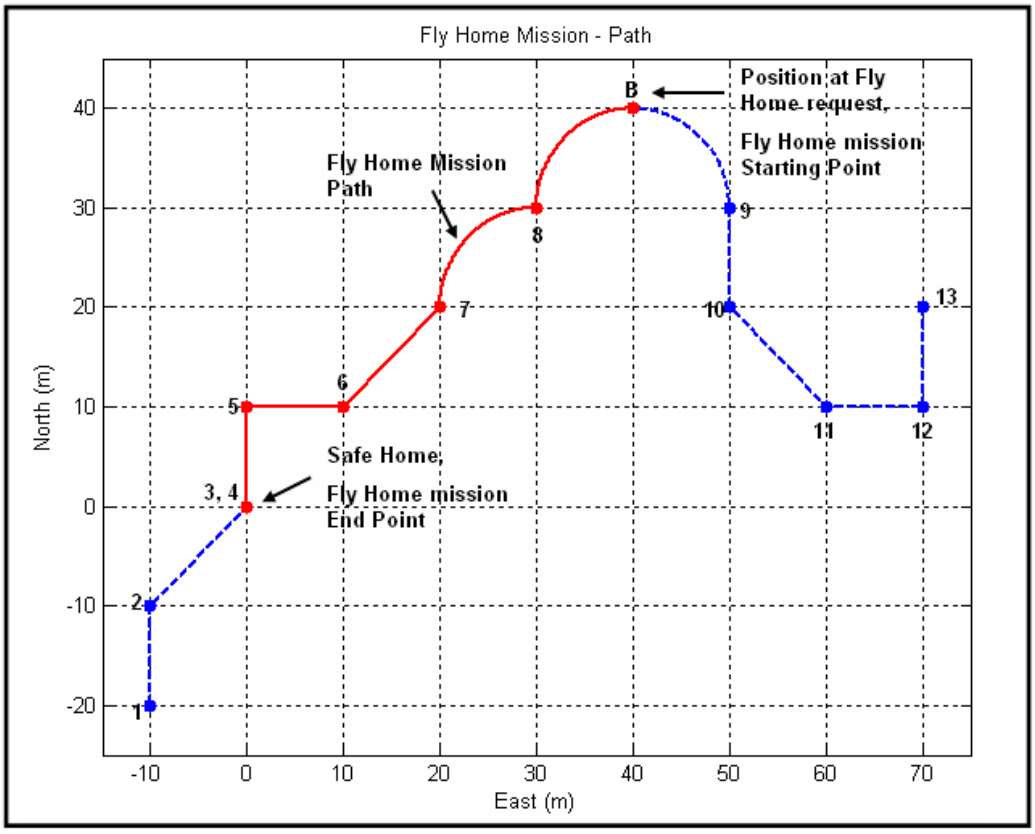

Figure 30: Fly Home mission's path, scenario 2

As illustrated in figure 30, the Fly Home mission starts on point B and ends on waypoint 4, the Safe Home of the mission. Furthermore, the path of the mission is such that it visits waypoint 8 till 4 sequentially and the path's points are conformant with the original mission's path. Moreover, other aspects of the Fly Home mission not shown in the picture, as shown on the previous subsection, have all been validated using Unit Testing. The waypoints before Safe Home position (waypoint 4) and after point B have not been visited by the Fly Home mission's path. The waypoint list of the mission generated by the Fly Home behavior can be found in the appendix.

It is important to point out that the Pirouette Around XY behavior in the original mission leading form waypoint 8 to waypoint 9 has the following parameters.

PI (30 [m], 40 [m], 10 [m/s], 180 [°])

The original mission is interrupted when the UAV was executing such behavior, and had completed 90° of its total angular displacement. Therefore, the first pirouette in the Fly Home mission was set to execute only the remaining part of the angular displacement, as shown in the figure above, and the behavior has the following parameters.

PI (30 [m], 40 [m], 10 [m/s], -90 [°])

4.5.1.3 Third Scenario

In the third scenario, the mission is executed from its starting behavior until the Hover To behavior leading from waypoint 10 to waypoint 11. During the execution of such behavior, a Fly Home request is received and the mission is interrupted. In such moment, the UAV's position is at point C = (15, 55, -6) [m]. The configuration of the system is set to define the Home position with Full Home definition. Therefore, the Home position in such circumstances is waypoint 1, unlike the cases shown on the two previous subsections.

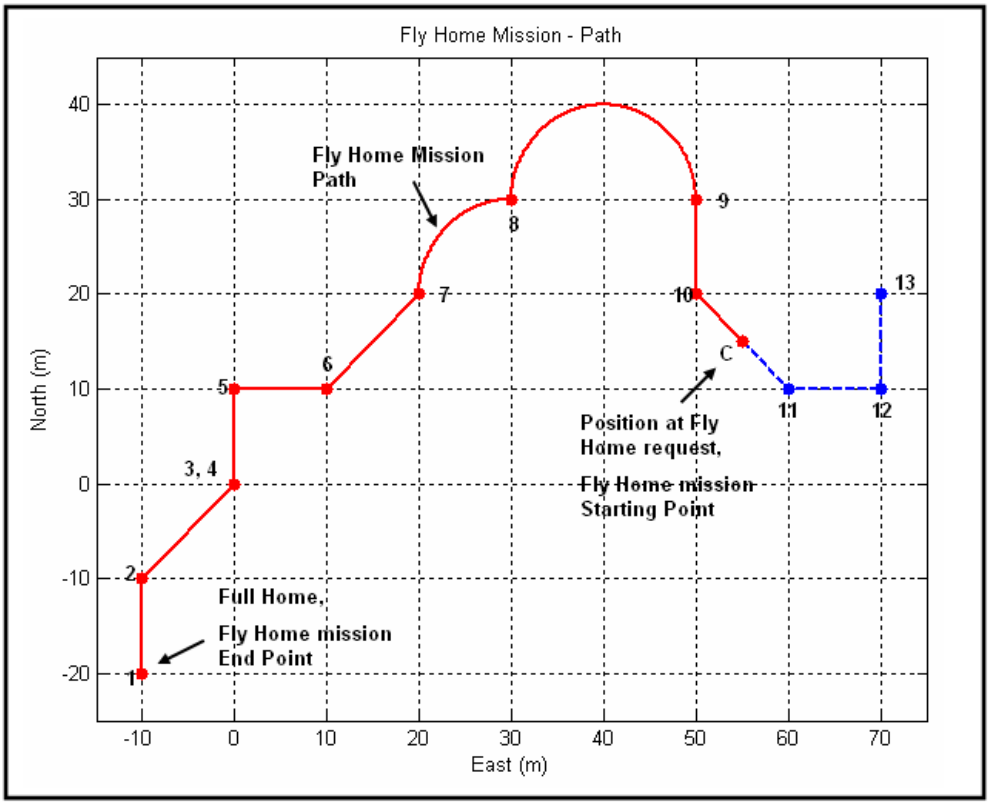

Figure 31: Fly Home mission's path, scenario 3

In such scenario, we applied the mission planning capabilities of the Fly Home behavior to plan a mission towards home. The path of the resultant mission is shown in figure 31.

As illustrated in figure 31, the Fly Home mission starts on point C and ends on waypoint 1, the Full Home of the mission. Furthermore, the path of the mission is such that it visits waypoint 10 till 1 sequentially and the path's points are conformant with the original mission's path. Moreover, other aspects, as enumerated on subsection **4.5.1.1**, of the Fly Home mission not shown in the picture have all been validated using Unit Testing. The waypoints after point C have not been visited by the Fly Home mission's path. The waypoint list of the mission generated by the Fly Home behavior can be found in the appendix.

4.5.2 Back to Home Mission and Behavior Management

To test the functionalities of the Fly Home behavior mission planning capabilities and its management functionalities, an original mission was planned offline and a scenario as described on subsection **4.4.1.2** was tested using SITL simulation. The scenario describes a situation in which the path of the original mission has a crossover such that two different segments of the path go through the mission's Home location, i.e. a *Back to Home* situation. The objective of such test is to verify if the Fly Home behavior is capable of detecting that it has achieved Home before finishing the mission, interrupting the Fly Home mission and returning to Home. All of these functionalities were designed to operate online while the UAV executes its mission.

The description of the original mission and the results of such test are shown in figure 32. The Fly Home request in such scenario was issued by the operator *after* the original mission had been executed till completion, and the system was configured to use Full Home definition. Therefore, the paths of the original mission and the Fly Home mission contain the same points. The traced line shows such points. The full line depicts the actual path followed by the UAV in the simulation during the Fly Home flight. The waypoint list representing the mission is available at the appendix.

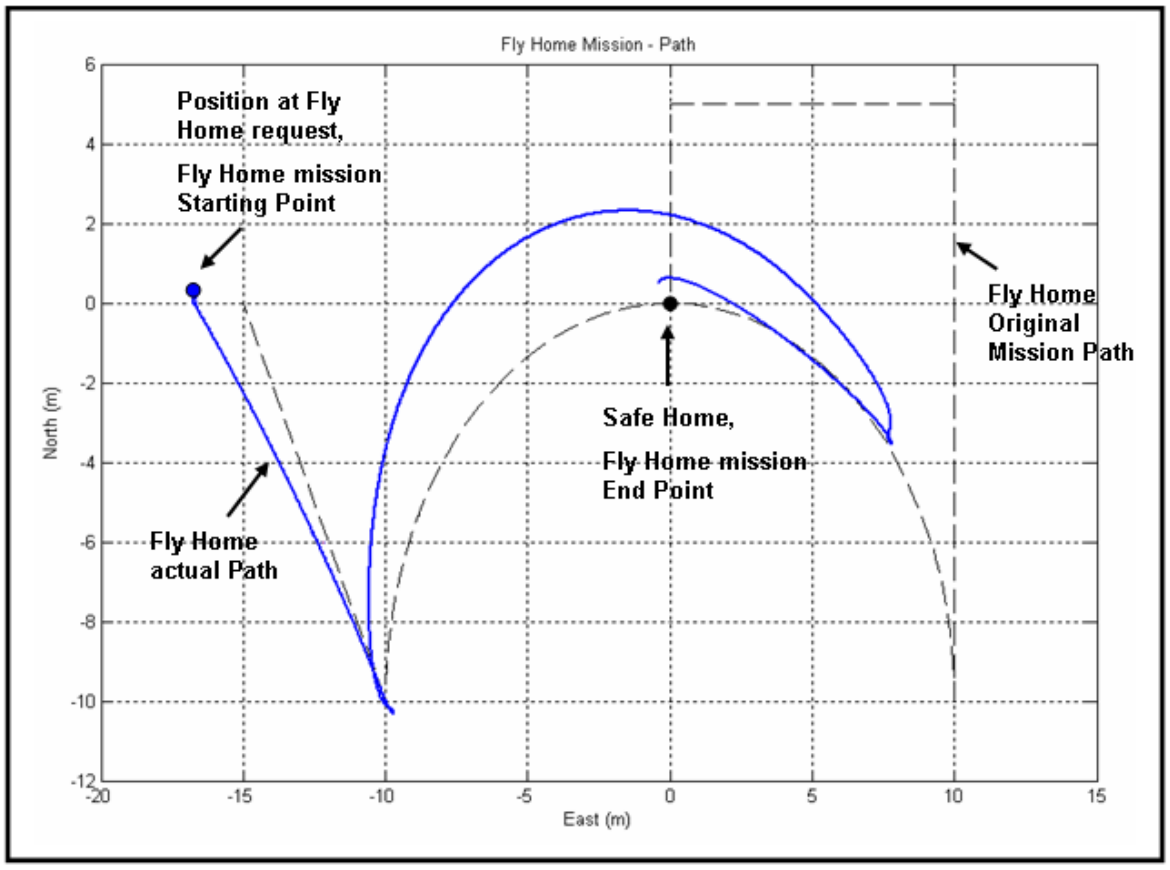

Figure 32: Back to Home mission

As shown in figure 32, the system recognized the proximity of the Home location before the Fly Home mission had come to an end. Then, the UAV was commanded to slow down, stopped around point (-3, 7.5, -6) [m], turned around and returned to Home location. By behaving in such manner, the system reached the Fly Home behavior's objective, i.e. reaching Home, without necessity of visiting waypoints such as (-10, 10, -6) [m], (5, 10, -6) [m] and (5, 0, -6) [m], resuming its mission earlier and thus enhancing the behavior's efficiency.

In this mission, the Fly Home behavior was also tested with regard to its management capabilities, since the test was realized online and the state transitions and processes must have been executed properly in order to secure the success of the test. Figure 33 presents the temporal evolution of the test.

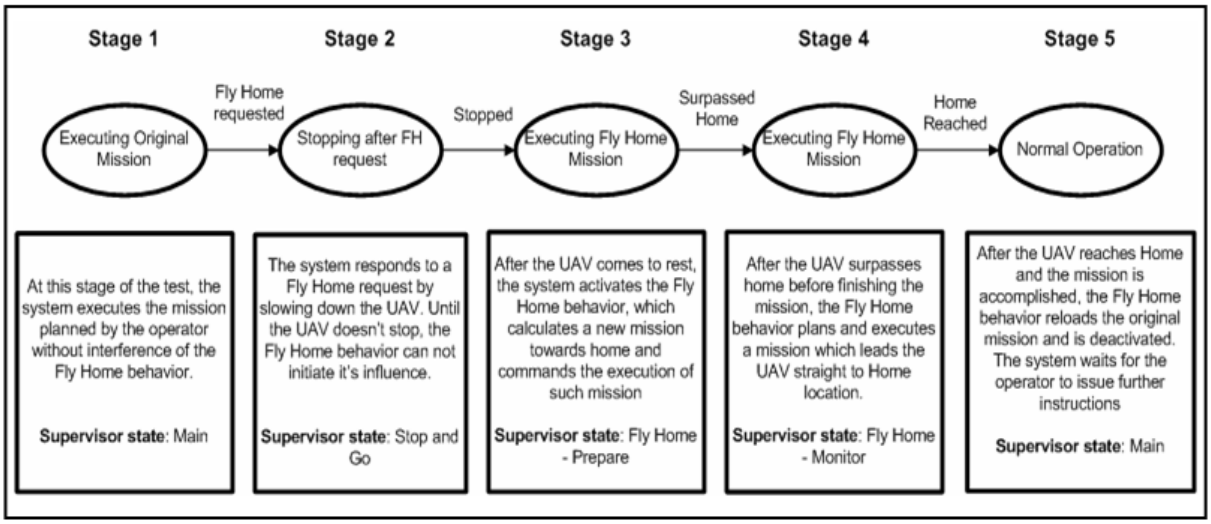

Figure 33: Temporal evolution of a Fly Home mission

The presented temporal evolution of the Fly Home mission was checked using SITL simulation and corresponds to the expected functionality of the behavior, as shown on section **4.4**.

5 Search Object

This chapter describes the intelligent behavior module designated *Search Object*. This module has been conceived, implemented and successfully tested.

The visual capabilities of ARTIS enable the system to perform missions to search and track a ground object of known shape [27]. In such missions, the sequence of tasks is planned offline by the operator. Normally, the part of the mission in which the UAV performs the search for the ground object is planned such that the UAV follows a path covering a search area defined by the operator [21], [28]. In case the UAV establishes visual contact with the ground object, the helicopter is commanded to try to follow the object and ceases to perform the tasks in the original mission. In case the visual computer of ARTIS does not accuse visual contact, the original mission is performed by the UAV till completion or interruption.

In the occasion when the UAV interrupts the original mission to try to follow the ground object, the movements of the helicopter are defined online. Therefore, once the UAV starts tracking the object, its effective trajectory is no longer based on the tasks planned by the operator.

The tracking of the ground object can be interrupted by several events, e.g. when visual contact with the object is lost for a certain period of time. Therefore, the UAV's mission is undefined, for the Object Tracking mission is over and the original search mission has been abandoned at the moment the system activated the Object Tracking behavior. In such circumstances, human operator assistance is required to plan the following tasks of the helicopter.

The potential capacity of the system to autonomously reset the original mission in a safe manner would grant the system a higher level of autonomy, since the necessary human intervention is hence reduced [23]. Furthermore, it is desirable that the system would be able to adapt its plans to conciliate the changes caused by the spatial dislocation during the following of the ground object. In other words, the system should be able to recognize which stage of the original mission is more compatible with the UAV's position at the moment. The development of the ability of the system to adapt its plans according to external changes (e.g. the UAV displacement caused by the following of the movement of the ground object) implies an enhancement on the autonomy of the system [22], [23].

This chapter presents a method used by the Search Object behavior to compute the area sectors in which the search mission is conducted and establishes a strategy to restart a

search mission considering the UAV's position in the context the mission itself. Furthermore, the modeling of the Search Object behavior using a statechart is presented as well as strategies used by the behavior to manage events and state transitions from the moment when the system initiates the search for the ground object till the moment when the system returns to its normal operation state.

5.1 Problem Statement

The Search Object intelligent behavior addresses the problem of managing a mission dedicated to conduct a visual search of a specified ground object. The Search Object behavior is intended to function cooperatively with the Object Tracking behavior. A mission which coordinates both of these intelligent behaviors will be forth referred to as a *Search and Track mission*. The ground area scanned by the UAV's camera while searching for the ground object will be referred to as *search area*.

In a mission described by a behavior sequence, the behaviors scheduled to be activated while the system is actively checking for recognition of the ground object will collectively be referred to as *Search Task*. It is important to point out that a Search Task may contain one or more behaviors. Moreover, the Search Task must not be confused with the Search Object behavior or the Search and Track mission. The Search Task comprehends the behaviors in the mission planned by the operator which are executed while the vision system tries to recognize the object. The Search and Track mission comprehends the coordinated actions taken when the vision system searches the object. The Search and Track mission contains the Search Task and the Tracking Task (section **6.1**), which is planned and executed online without assistance from the operator. The Search Object behavior is the module which manages the Search Task and the fashion in which its behaviors are executed. Figure 34 illustrates a Search Task and how the Search Object and Object Tracking behaviors act during a Search and Track mission.

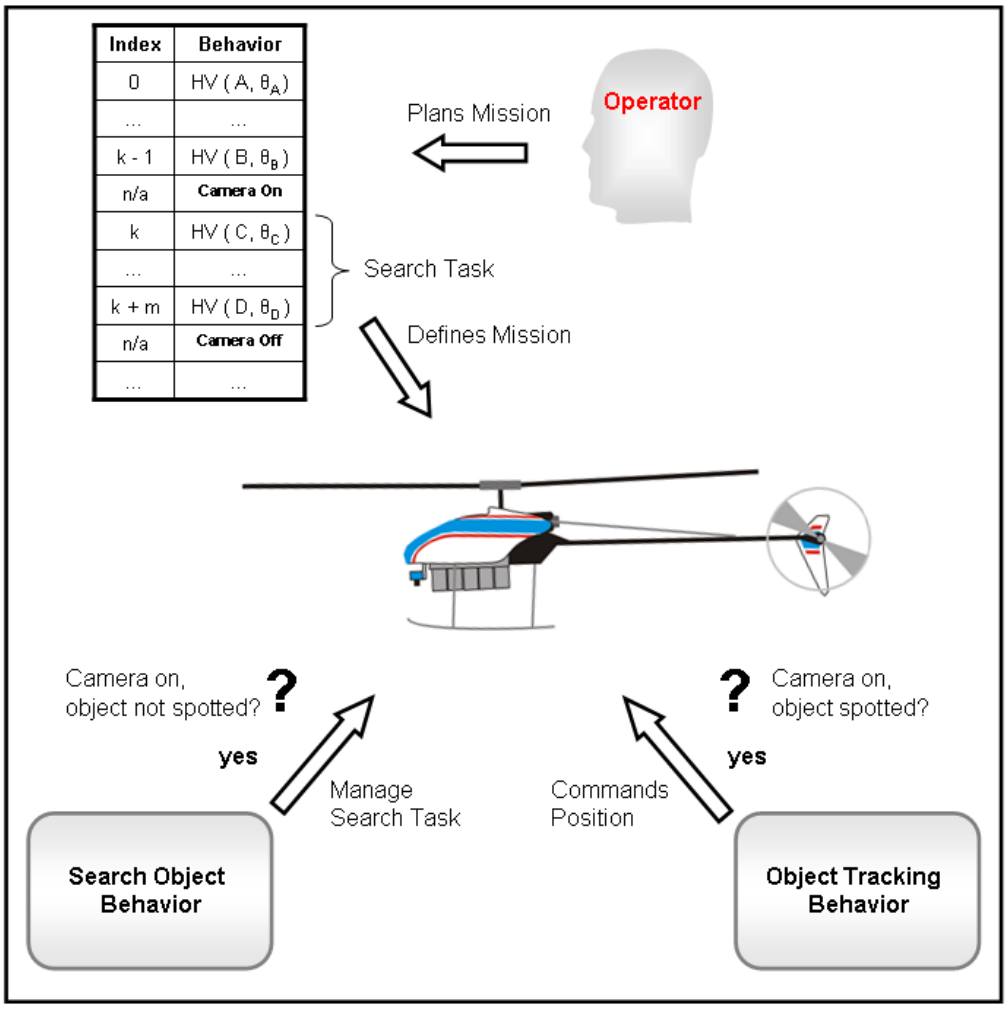

Figure 34: Search Task, Search Object and Object Tracking behaviors

Some directives were stated in order to establish the requirements of the Search Object behavior. These directives were based on desirable aspects of the conduction of a Search and Track mission and define the overall aspect of this problem. Figure 35 illustrates such aspects.

In all parts of figure 35, the search path planned by the operator is shown. In part one, the UAV initiates the Search and Track by flying through the search path. This is conducted by the embedded system following the behavior sequence which describes the mission. In beginning of this stage, the Search Object behavior initiates its operations. In part two, the UAV establishes visual contact with the ground object. The Search Object behavior ceases to operate, while Object Tracking behavior initiates its actions. In part three, the UAV tries to follow the target while operating under influence of Object Tracking behavior. In part four, the UAV looses visual contact with the target. The Object Tracking behavior ceases its activities while Search Object behavior is activated and conducts the UAV back to the closest segment of the search path in order to restart the search task. The object might also be

considered to be lost when it or the UAV flees the search area perimeter. In part five, the search task recommences through the original search path planned by the operator.

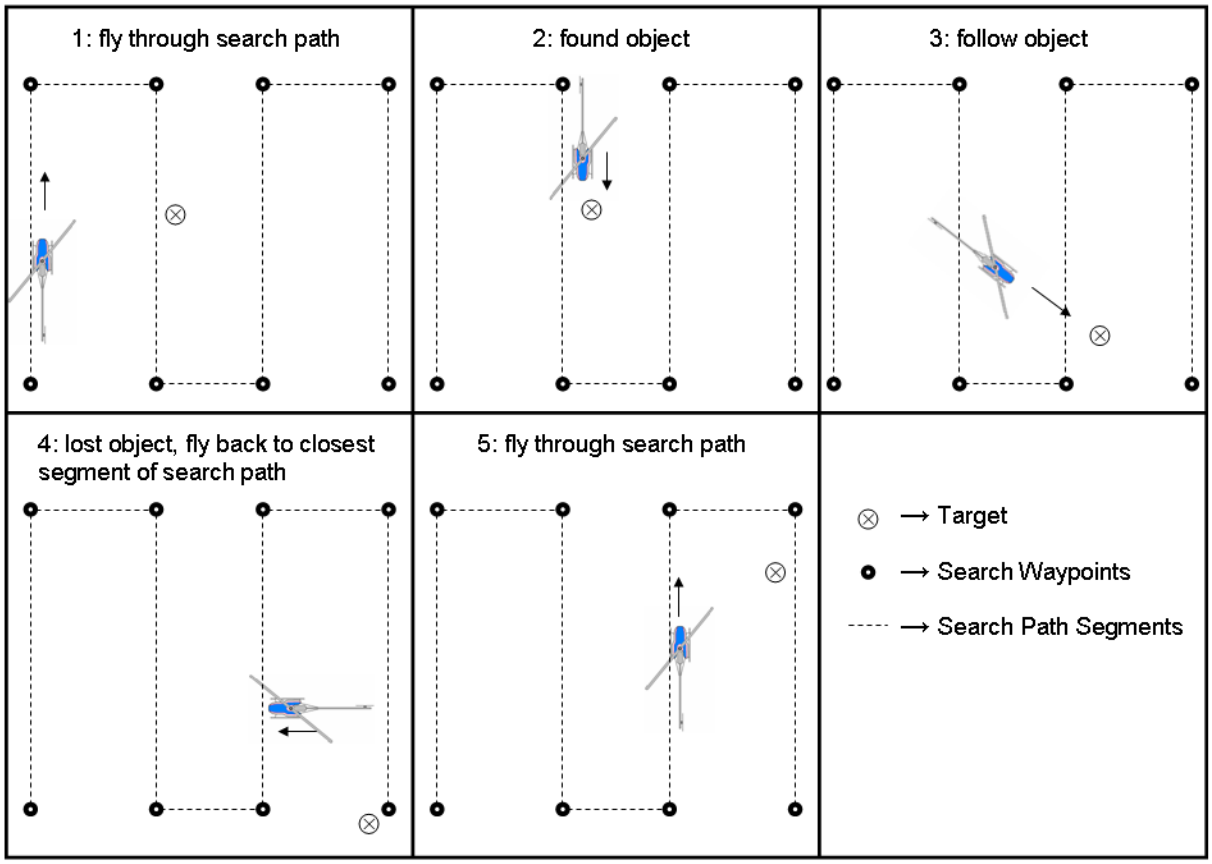

Figure 35: Search and Track mission

The directives which emerge from the desired aspects of a Search and Track mission are shown ahead.

- The only information necessary to achieve the functionalities of the Search Object behavior should be the behavior sequence describing the mission planed by the operator.
- All the involved computation in a Search Object mission should be done online without assistance external to the embedded system.
- The Search Object behavior should be able to recognize through the behavior sequence if a search task occurs during the course of the original mission.
- The Search Object behavior should be able to compute the perimeter of the searched area using the information of the behavior sequence about the waypoints contained in the search area. This perimeter is useful if the operator decides that the Search and Track mission is not to be executed outside of the Search area.

- The Search Object behavior should be able to identify situations in which a search task is implicit in the behavior sequence but it is not possible to calculate the searched area's perimeter (e.g. if the UAV is designated to search for the object in a straight line).
- If the object is found and tracked, the Search Object behavior should be able to record the original mission and reset it properly when the UAV stops tracking the ground object and it is desirable to restart the search.
- If the search task must be restarted, the UAV should return to the search path before recommencing the search.

5.2 Discussion

The aspects of the Search and Track mission under responsibility of the Search Object behavior can be separated into three sub-problems: calculating the perimeters of the area sectors in which the system searches for the ground object, restarting efficiently a search task and designing the strategies to safely manage all the possible events and steps of the procedure which could occur while the system is under the influence of the Search Object behavior.

To calculate the perimeters of the search area sectors, the behavior module could use the behavior sequence in order to extract the waypoints visited by the UAV while searching the object, as shown in subsection **4.3.1**. The mission planner separates the search area into convex cells; the problem of calculating the perimeter of each convex cell based on the waypoints contained in it can be solved by calculating the convex hull of the waypoints, a problem well studied in literature [29], [30], [31], [32], [33], [34], [35]. The approach proposed to solve the problem of calculating the perimeter of the search area sectors (including disconnected sectors) is to calculate the convex hull of the convex cells of each sector.

The nature of the problem of restarting efficiently a search task can be better analyzed with the assistance of the following question: in which stage of the original search mission should the system recommence the mission? The stages of the mission, as described before, are defined by the behaviors in the behavior sequence. To recommence the mission, it is enough to choose the first behavior to be activated when original behavior sequence is reloaded to the Sequence Control module [9].

It is important to point out that the mission is considered to be restarted only when the UAV's path coincides again with the path planned by the operator. Therefore, it is desirable that the UAV is back to the original path as quickly as possible.

Considering these aspects, the approach used in this work to address the problem of efficiently resetting the search mission is using the original mission to calculate which point[†] of the original path is the closest to the UAV's position, flying to this point and then resetting the original behavior sequence with the behavior corresponding to the path segment containing the closest point.

The approach proposed to develop the Search Object behavior's management capabilities is to model the behavior using Statechart diagrams for the reasons presented in the modeling of the Fly Home behavior.

5.3 Implementation

In order to successfully meet the directives proposed for Search Object behavior, it is required to define methods to compute the necessary entities. The perimeter of the search area must be computed based on the information contained in the behavior sequence. Also, to reset a search mission, the behavior must be able to calculate and compare the distance between a point (in this case, the UAV's position) and segments of the search path. Furthermore, the point which minimizes the distance between a path segment and the UAV's position must be calculated. This section presents and explains several aspects that must be addressed in the scope of the Search Object mission, as well as the solutions proposed.

5.3.1 Search Area Perimeter

This subsection addresses the problem of defining and computing the search area's perimeter of a mission. Since the scope of the problem involves areas defined in $\mathbf{R}^2$ space, all the waypoints will be analyzed solely by their x and y coordinates. Therefore, in this subsection we will implicitly refer to waypoints as points in $\mathbf{R}^2$.

[†] Not to be confused with a waypoint of the path. This point could be located in the path segment connecting two consecutive waypoints.

5.3.1.1 Defining the Perimeter

Consider a Search Task T_{search} with n behaviors designating m waypoints, $n, m \in \mathbf{N}$, following the restriction $n \geq m$ (i.e. each behavior designates one waypoint or none). Let $P \subset \mathbf{R}^2$ denote the set of points of the search area perimeter. Let p: I $\rightarrow P$ denote the continuous map defining the path of the perimeter, with I denoting the unit interval [0,1]. Note that the initial point and terminal point of the path are, respectively, $p(0)$ and $p(1)$. Let $W \subset \mathbf{R}^2$ denote the set of m waypoints.

Before evaluating the possible solutions on how to define and calculate the search area perimeter, it is important to establish forehand the restraints to which the solution candidates must comply. Three intuitive restraints were identified. Figure 36 describes visually these restraints.

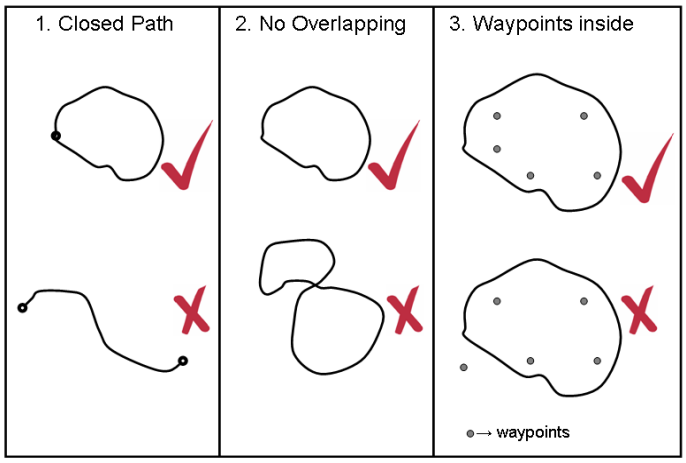

Figure 36: Perimeter Restraints

1. The perimeter must be closed;
2. The perimeter must not present overlapping;
3. Every waypoint of W must be either inside or on the perimeter.

The formal description of such restraints is presented in (4-1), (4-2) and (4-3).

Restraint 1: (5-1)
P describes a valid search area perimeter $\Rightarrow p(0) = p(1)$

Restraint 2: (5-2)

$$P \text{ describes a valid search area perimeter} \Rightarrow \nexists\, t_1, t_2 \in (0,1) \,\big|\, t_1 \neq t_2 \text{ and } p(t_1) = p(t_2)$$

Restraint 3[†]: (5-3)

Let $w = (x_w, y_w) \in W$ be the test point;
Let Q be a subset of P such that $q = (x_q, y_q) \in Q \Leftrightarrow y_q = y_w \wedge y_q > y_w$;
Let n_Q be the number of elements of set Q;

P describes a valid search area perimeter $\Rightarrow$
$\forall w = (x_w, y_w) \in W$, $w \in P \vee n_Q$ is an odd integer;

With these restraints defined, it is possible to identify valid candidates for the perimeter of the search area. The candidates are proposed according to assumptions made over the Search Task. In case one of the assumptions is not reasonable on a search object mission, the operator has the authority to disable the Search Object behavior and proceed with the mission without the behavior's management. Below follows the list of assumptions.

1. The search mission was planned such that the operator intends to search the object on a specified area, and not a path;
2. The path designated by the Search Task is intended by the operator to cover the search area;
3. Every waypoint designated by the path of the Search Task is within the search area;
4. The operator sets waypoints on the borders of the search area in order to guarantee that the area is searched completely.
5. The Search Task is planned by the operator to generate a search area separated into convex cells.

The assumptions are all in compliance with the missions generated by the modules responsible to compute the search area missions in the ARTIS project [21]. Figure 37 illustrates a search area mission planned with the assistance of software MAESTRO.

[†] The formal description of Restraint 3 is based on the Jordan Curve Theorem presented on the subsection 6.4.1. In case the test point is aligned with an edge of the perimeter, i.e. Q has an infinite number of elements, the adaptation presented on subsection 6.4.1 is applied, maintaining the validity of the test.

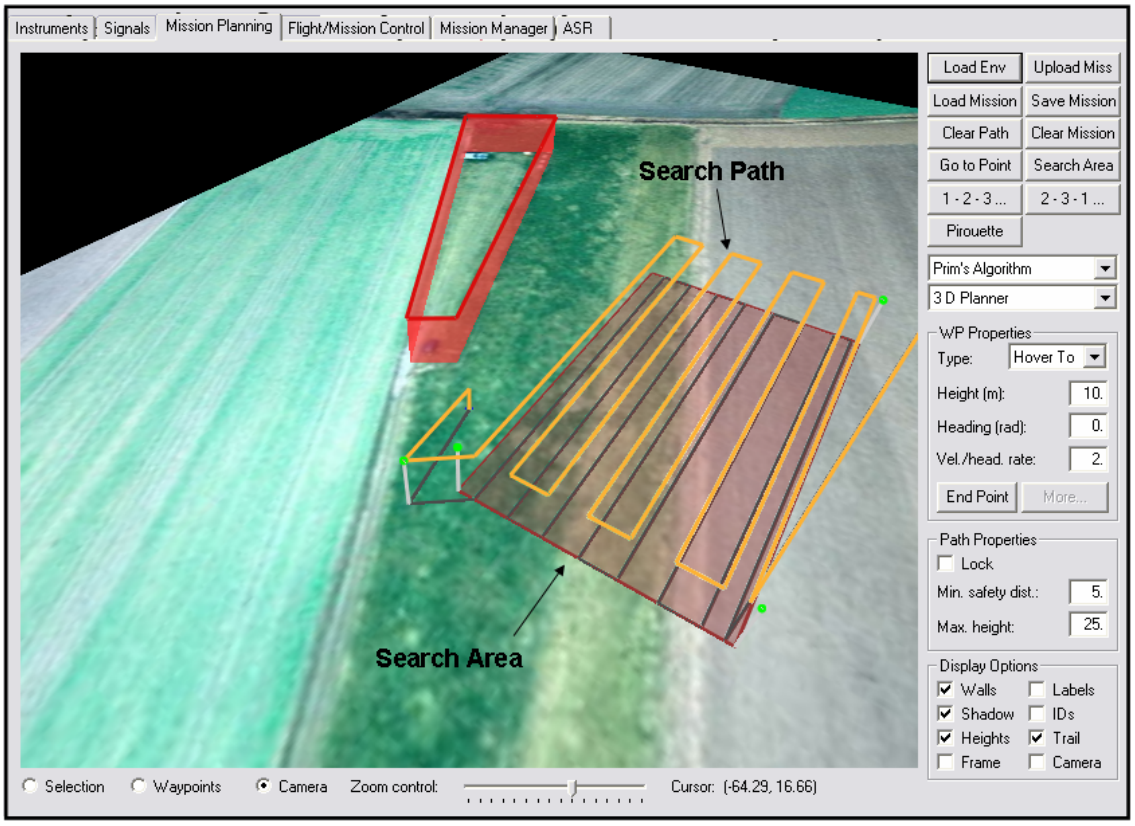

Figure 37: Search area mission example. The search area is entered by the operator, while the search path covering such an area is computed by Maestro.

The solution candidates for this problem must comply with the restraints and the assumptions made. One possible candidate would be finding the *convex hull* of each convex cell of the search area defined by the set of waypoints *W*. In (5-4) and (5-5) we define the convex hull for the vector space $\mathbf{R}^2$, according to the scope of the addressed problem.

The convex hull of a set *C* in $\mathbf{R}^2$ may be visualized by imagining an elastic band stretched around the set of points; once released, the elastic band encompasses every point of *C* while touching only the points belonging to the convex hull.

Definition 3[†]: (5-4)

Let *C* be a set in $\mathbf{R}^2$. *C* is said to be *convex* if:

$$\forall x_0, x_1 \in \mathbf{R}^2, \forall t \in [0,1]$$
$$x_0, x_1 \in C \Rightarrow (1-t) \cdot x_0 + t \cdot x_1 \in C$$

[†] http://en.wikipedia.org/wiki/Convex_set

Definition 4[†]: (5-5)

The *convex hull* for a set of points X in a $\mathbf{R}^2$ is the minimal convex set containing X.

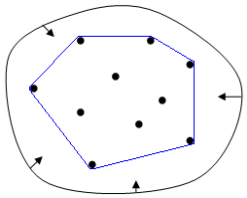

Figure 38: Convex hull, elastic band analogy[‡]

Some characteristics of the convex hull contribute to its selection as perimeter representation. First, it complies with the restraints and assumptions declared previously. Second, the convex hull not only encloses not only every waypoint but also every straight path connecting the points. Third, a "smaller" perimeter would either not be convex or not contain every waypoint. Fourth, a "bigger" perimeter would possibly enclose regions not covered by the search area defined by the operator.

In the following subsection we address the problem of calculating the convex hull of a set of waypoints in vector space $\mathbf{R}^2$.

5.3.1.2 Computing the Perimeter

Several algorithms were developed to solve the problem of computing the convex hull of a set of points in a vector space [36]. The specific planar case (i.e. set of points in vector space $\mathbf{R}^2$) has also been addressed by several authors, and a number of algorithms have been proposed [29], [30], [31], [32], [33], [34], [35]. Table 3 relates algorithms proposed and their time complexity. In the table, n refers to the number of points in the input set of the problem and h refers to the number of vertices on the output convex hull.

Due to implementation simplicity, time complexity and characteristics to be shown ahead, the algorithm chosen to compute the perimeter of a Search and Track mission was the Monotone Chain algorithm [29].

[†] http://en.wikipedia.org/wiki/Convex_hull
[‡] http://upload.wikimedia.org/wikipedia/commons/b/bc/ConvexHull.png

Table 3: Convex Hull algorithms[†]

Algorithm	Time Complexity
Grahan Scan [33]	$O(n \cdot \log(n))$
Jarvis March [32]	$O(n \cdot h)$
QuickHull [34]	$O(n \cdot h)$
Divide-and-Conquer [31]	$O(n \cdot \log(n))$
Monotone Chain [29]	$O(n \cdot \log(n))$
Incremental [35]	$O(n \cdot \log(n))$
Marriage-before-Conquest [30]	$O(n \cdot \log(h))$

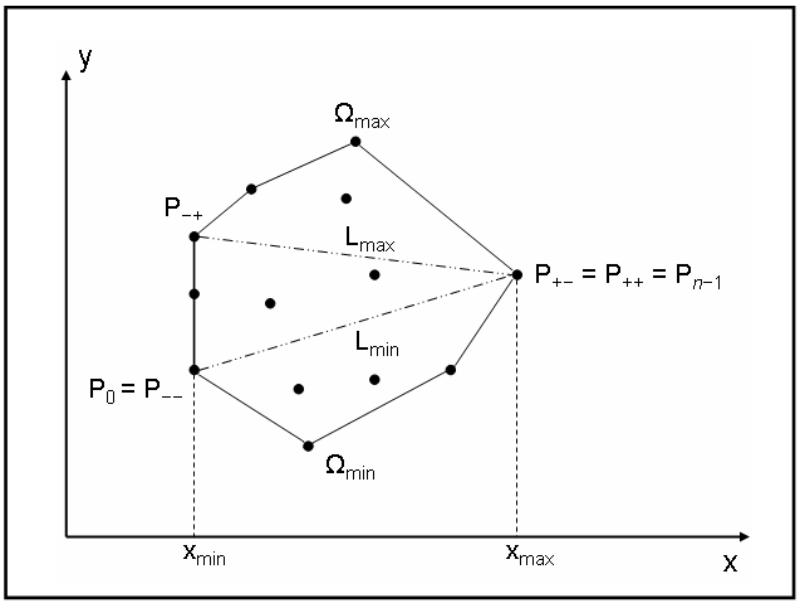

Figure 39: Monotone Chain

Consider the problem of calculating the convex hull of a set of n planar points[†]. The first step of the Monotone Chain algorithm is to sort the point set $S = \{ P_0, P_1, \ldots, P_{n-1} \}$ by increasing x and then y coordinate values of the points. The minimum and maximum x coordinate values will be referred to respectively as x_{min} and x_{max}. Hence, $P_0.x = x_{min}$, where the notation $P_i.x$ refers to the value of the x coordinate of point P_i. Let P_{--} be the point with $P_{--}.x = x_{min}$ and minimal y coordinate value. Let P_{-+} be the point with $P_{-+}.x = x_{min}$ and maximal y coordinate values. If there is only one point with x coordinate value x_{min}, then $P_{--} = P_{-+}$. Let

[†] http://www.softsurfer.com/Archive/algorithm_0109/algorithm_0109.htm#References
[†] The algorithm description is based on the explanation found on http://www.softsurfer.com/Archive/algorithm_0109/algorithm_0109.htm#References

P_{+-} be the point with $P_{...}x = x_{max}$ and minimal y coordinate value. Let P_{-+} be the point with $P_{++}.x = x_{max}$ and maximal y coordinate values. If there is only one point with x coordinate value x_{max}, then $P_{+-} = P_{++}$. Next, we connect points P_{--} and P_{+-} with a straight line named L_{min}. Similarly, we connect P_{-+} and P_{++} with a straight line named L_{max}. These entities are shown in figure 39.

The algorithm calculates the convex hull by calculating Ω_{min} and Ω_{max}. Ω_{min} is the lower convex chain below L_{min} and the line connecting P_{--} and P_{-+}. Ω_{max} is the upper convex chain above L_{max} and the line connecting and the line connecting P_{+-} and P_{++}. The convex hull is constructed by joining Ω_{min} and Ω_{max} together.

The algorithm uses a stack of points to calculate the convex chains. For the lower chain, P_{--} is the first point stacked. Then the points of S are sequentially processed, but considering only the points below L_{min}. Suppose that at a stage of the algorithm, the lower convex chain has been calculated by processing every point of S before P_k. If P_k is below L_{min}, it is processed. If the only point in the stack is P_{--}, P_k is pushed in the stack and the next iteration occurs. Otherwise, let us name the top point of the stack P_{-1} and the second top point P_{-2}. If P_k is strictly left of the line from P_{-2} to P_{-1}, P_k is pushed into the stack. Otherwise, pop P_{-1} off the stack and test P_k once again. Proceed until P_k gets pushed into the stack. The points in the stack represent the lower hull for the points already processed. After all points have been processed, push P_{+-} onto the stack. The stack represents the vertices of the lower convex chain Ω_{min}.

The upper convex chain Ω_{max} is constructed analogously, processing S in decreasing order from P_{n+1} to P_0, starting the stack with P_{++} and considering only the points above L_{max}. The convex hull of the given set of points is the junction of the convex chains Ω_{min} and Ω_{min}. A pseudo code for the Monotone Chain algorithm is presented on the appendix.

One characteristic which contributes to the performance of the Monotone Chain algorithm within the scope of ARTIS is the fact that in case there is alignment of points in the values x_{min} and x_{max}, exists the possibility that some points will not be processed, and therefore the computation time decreases. This situation occurs often in missions planned by the Search Area modules of ARTIS [21], [28].

5.3.2 Analysis of Search Mission

Before computing the perimeter of the Search Task, it is necessary to specify the criteria utilized to verify if a valid perimeter could be extracted from the behavior sequence describing a mission.

The first criterion concerns the use of the camera during the mission. If the mission does not explicitly designate the use of the camera during its course, we assume that the operator did not intend the mission to include a Search Task. This could be checked on the behavior sequence by verifying the presence of behaviors indicating a command to turn the camera on.

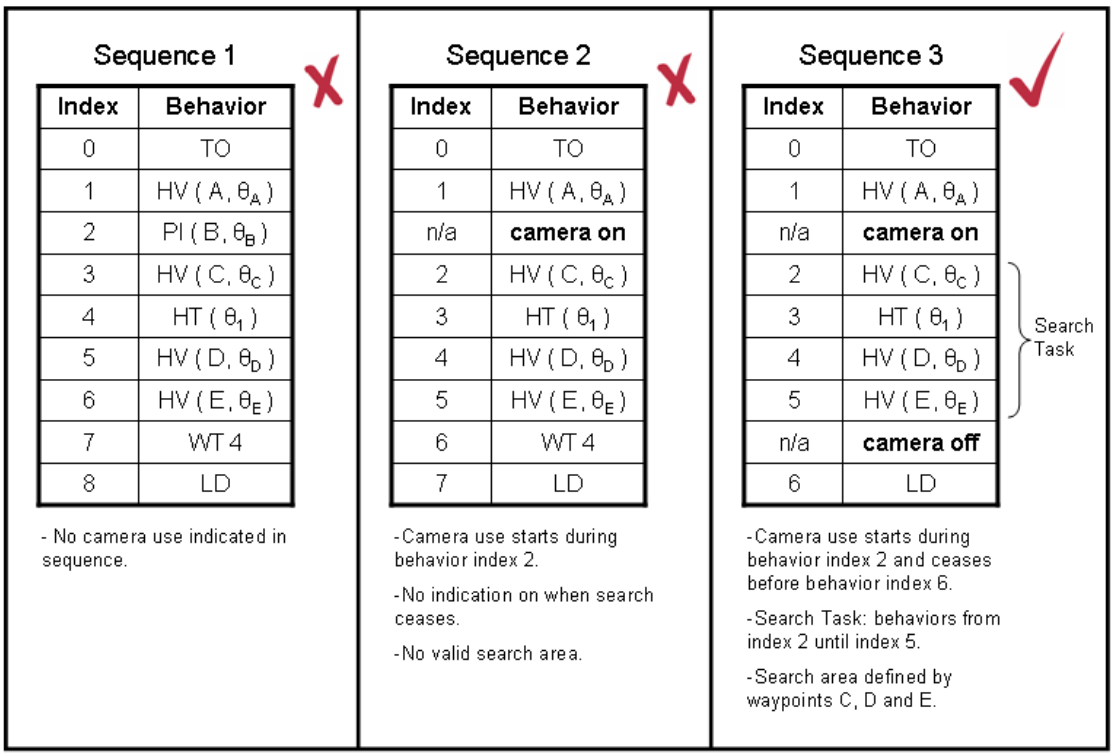

Figure 40: On the possibility of computing the search area perimeter of a mission

The second criterion regards initiating and finishing properly a Search Task. If the mission specifies that the camera should be turned on at some point of its course and that it should be turned off after the completion of a number of tasks, it is assumed that the waypoints visited during this period define a cell of the search area planned by the operator. If this assumption is not reasonable in a given mission, the operator has the authority to signalize that the mission does not define a search area through the system's configuration

settings. Figure 40 exemplifies the application of these criteria to check if a mission provides a valid set of waypoints to calculate the search area perimeter.

The set of points required to calculate the search area perimeter are provided by the waypoints designated by the position behaviors contained by the Search Task. In the example provided by *Sequence 3* in the figure above, the set of points would be defined by W = {C, D, E }.

It is important to point out that, in our approach, having a set of tasks preceded and finished by, respectively, a *camera on* and a *camera off* switch is a necessary but not sufficient condition to extract a search area perimeter from the behavior sequence. The other condition is that the tasks in such set include position behaviors which designate at least three non-collinear waypoints. Figure 41 illustrates such condition.

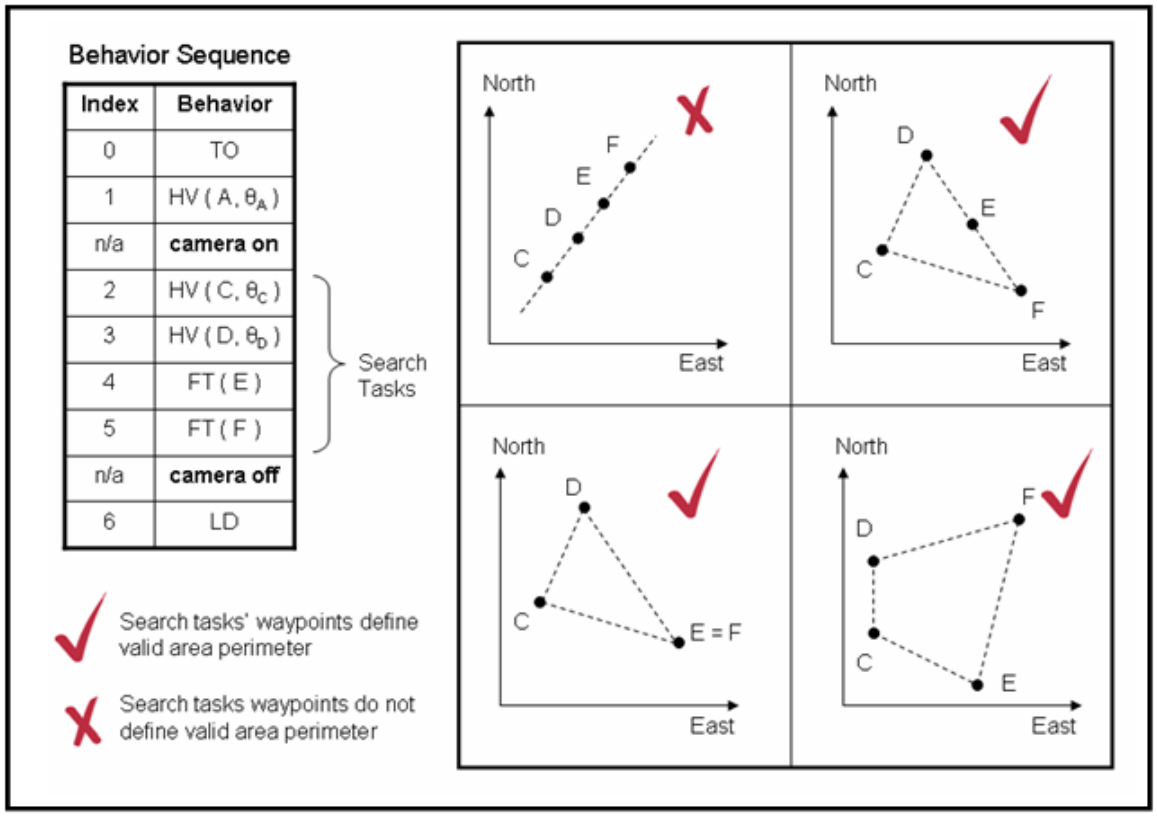

Figure 41: Perimeter computation possibility according to search tasks' waypoints spatial disposition

5.3.3 Computing Concave and Disconnected Perimeter Sectors

The search missions planned by the off-board program MAESTRO do not always comprise simply single convex areas. As stated before, in several search missions the search is divided into several search areas which are not always convex. In such cases, the method to

calculate the search area perimeter as shown in subsection **5.3.1** does not provide a satisfactory solution, since its hypotheses are violated. It is necessary then to extend the method presented.

Each search area on the mission might not be convex. However, it is possible to decompose the concave search area into convex cells [37] and use the algorithm explained before to calculate the perimeter of each individual cell. The separation of the search area into convex cells is also used by ARTIS' mission planner to plan search missions [21].

The identification of each individual search cell in the behavior sequence is possible using the same approach shown in the previous subsection. The missions in such conditions are purposely planned such that the set of tasks realized in each search area cell is preceded by a *camera on* switch and finished with a *camera off* switch. Figure 42 exemplifies a mission comprehending two distinct convex search area cells.

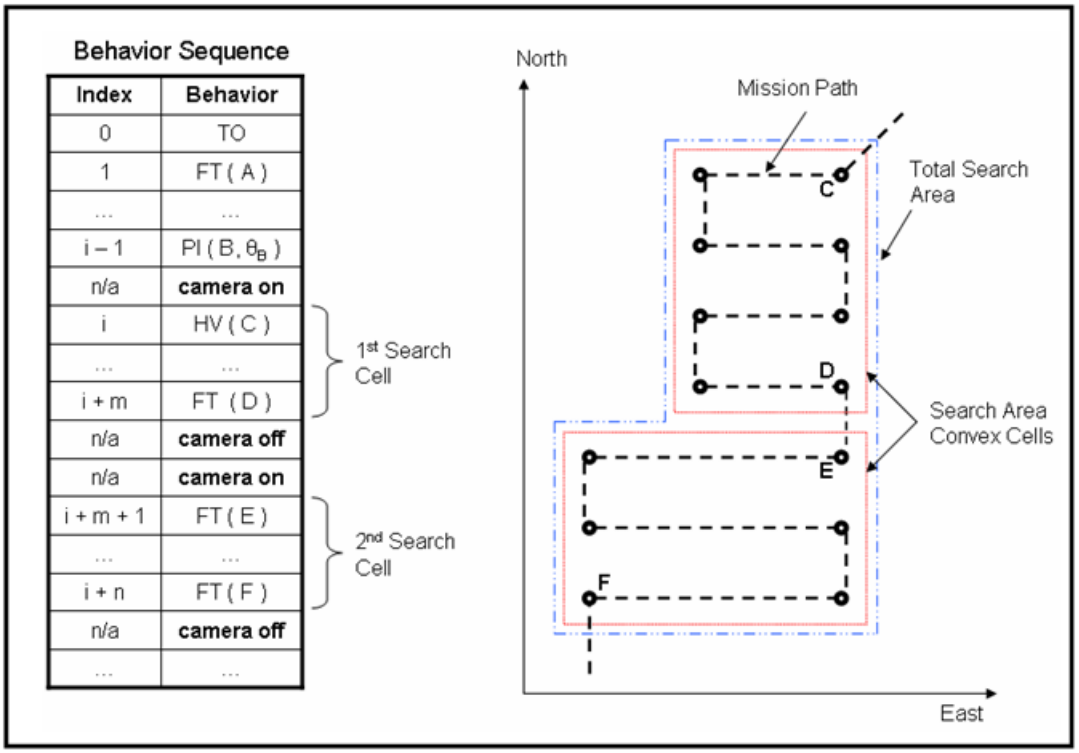

Figure 42: Search cells in the behavior sequence

The approach used to solve this problem consists in calculating and storing each individual convex cell of the complete mission. The concept is illustrated on figures 43, 44 and 45.

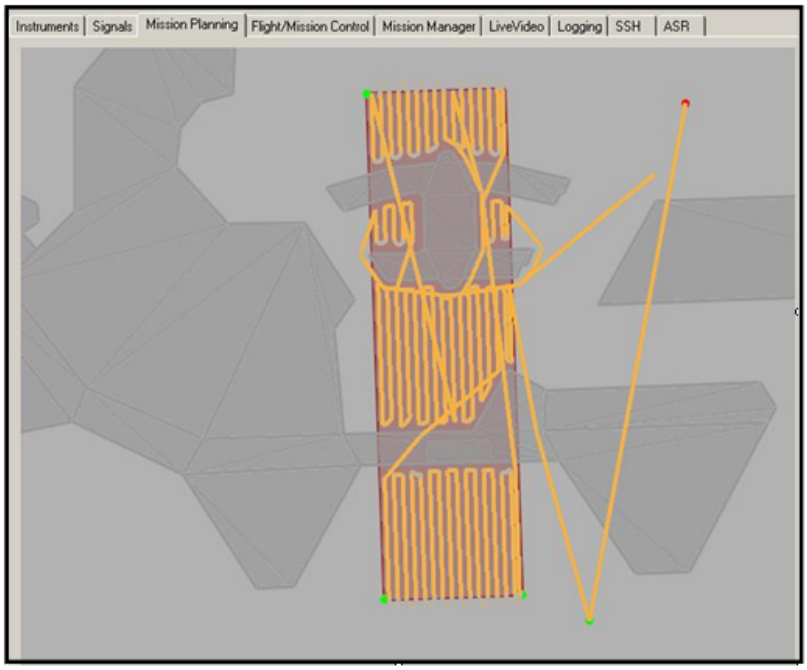

Figure 43: Plan of a search mission around a model of the Hannover Airport. The dark gray area represents obstacles, while the light gray area represents free space. The search area requested by the operator is represented in the middle rectangle, and the tracks represent path planned by Maestro. Path segments outside the search area represent paths connecting other waypoints outside the search area required to be visited by the operator.

Figure 43 describes the planning of a search mission around a model of the Hannover Airport. The total search area designated by the operator can be recognized in the figure as the red area under the mission path. To execute the search properly, the mission planner separates the total area into convex cells in order to comprehend all the intended search area sectors which are free from obstacles.

Figure 44 shows the path of the search mission as extracted by the Search Object behavior from the behavior sequence representing such mission. The calculation of the mission's path is executed using the same principles explained before on subsection **4.3.1**.

Figure 45 presents the perimeter convex cells calculated by the Search Object behavior based on the search mission's behavior sequence. As shown, the convex area cells in which the UAV proceeds with the search are separated and each of their convex perimeters is computed using the methods described in subsections **5.3.1** and **5.3.2**.

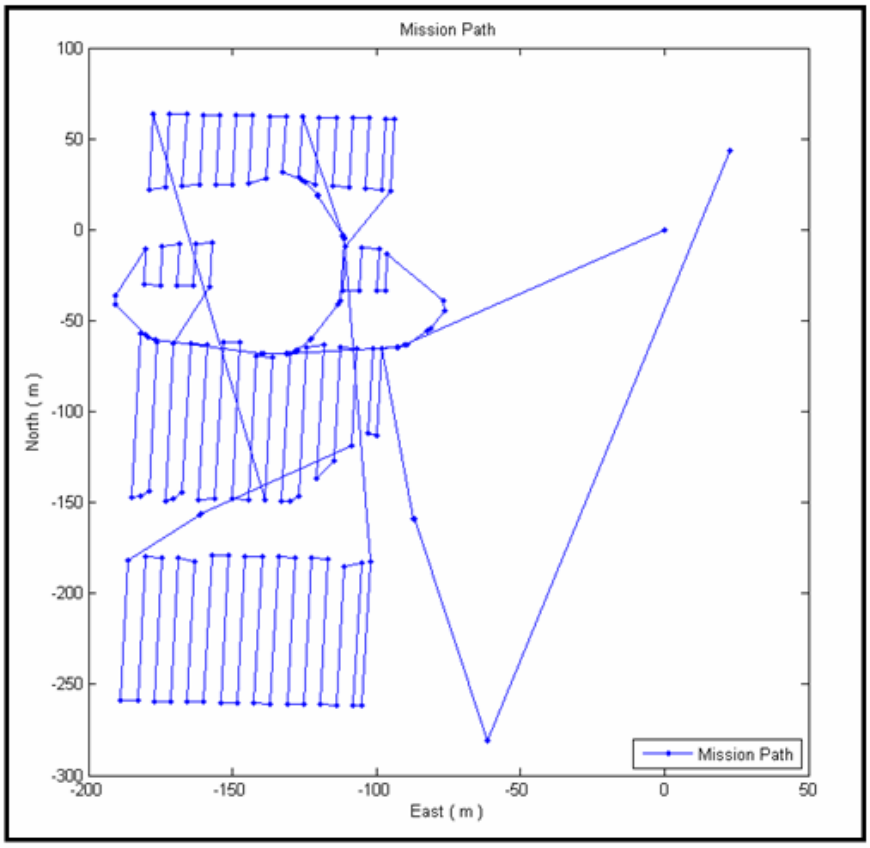

Figure 44: Search mission's path

5.3.4 Reset Search Mission

As shown in figure 35, one of the characteristics of the Search Object behavior is returning to the search path after a failed Object Tracking behavior. In this subsection, we address three problems:

1. Finding the closest path segment to the UAV's position;
2. Determining the closest point of a given path segment relative to the UAV's position;
3. Assemble the mission leading the UAV back to the search path.

The first problem can be solved with simple vector algebra theory. The distance from each path segment to the UAV's position can be calculated by the formula in (5-6). The demonstration is presented on the appendix.

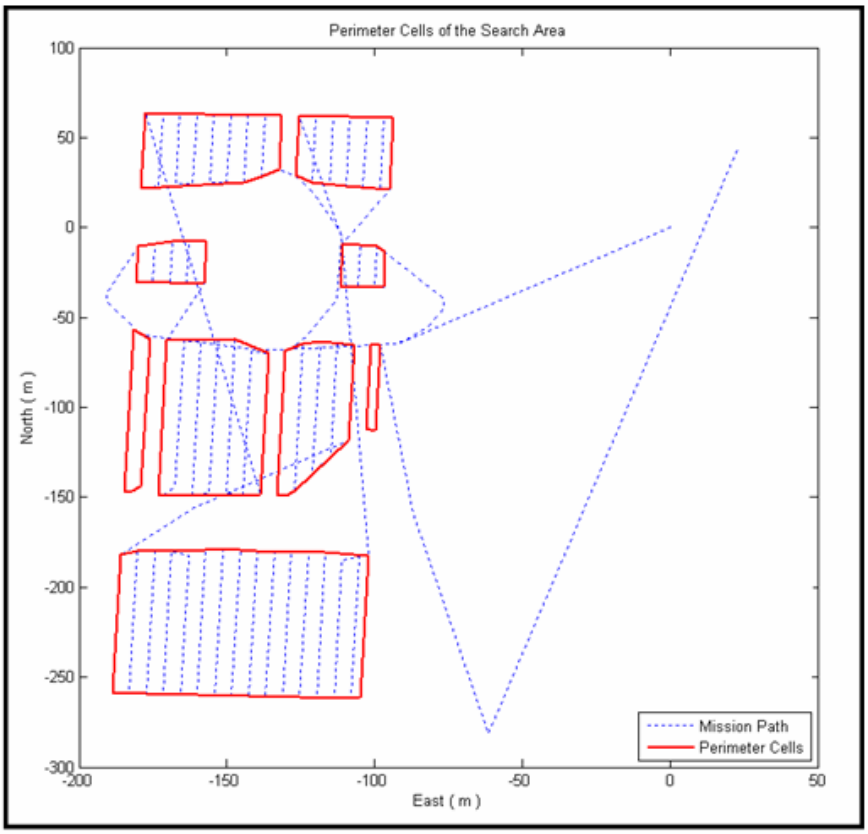

Figure 45: Perimeter cells from the search mission's area

Therefore, in order to find the closest path segment to UAV's position, we calculate the distance between the UAV and each path segment of the search tasks of each search cell and choose the segment which minimizes such distance.

Formula 1: (5-6)

$$\text{Point A}\,(x_a,y_a)\in\mathbf{R}^2$$

$$\text{Segment B}\,(x_b,y_b),\text{C}\,(x_c,y_c)\in\mathbf{R}^2$$

$$d=\begin{cases}\sqrt{(x_a-x_b)\cdot(x_a-x_b)+(y_a-y_b)\cdot(y_a-y_b)}\,, & \text{if } (x_a-x_b)\cdot(x_c-x_b)+(y_a-y_b)\cdot(y_c-y_b)<0\\ \sqrt{(x_a-x_c)\cdot(x_a-x_c)+(y_a-y_c)\cdot(y_a-y_c)}\,, & \text{if } (x_a-x_c)\cdot(x_b-x_c)+(y_a-y_c)\cdot(y_b-y_c)<0\\ \dfrac{(x_b-x_a)\cdot(y_c-y_a)+(x_c-x_a)\cdot(y_b-y_a)}{\sqrt{(x_b-x_c)\cdot(x_b-x_c)+(y_b-y_c)\cdot(y_b-y_c)}}\,, & \text{else}\end{cases}$$

The second problem, calculating the closest point of the closest path segment, can also be solved using simple vector algebra theory. Mathematically, the problem resumes in computing the closest point P of a segment BC with respect to point A in the plane. The point can be calculated by the formula in (5-7). The demonstration is presented on the appendix.

Formula 2: (5-7)

$$\text{Points } A(x_a,y_a), B(x_b,y_b), C(x_c,y_c) \in \mathbf{R}^2;$$

$$\text{Let } m = \frac{y_c - y_b}{x_c - x_b},\ a = \frac{x_c y_b - x_b y_c}{x_c - x_b},\ BC = \{(x,y) \in (x_b,y_b) \times (x_c,y_c) \mid y = mx + a\};$$

$$\text{Point } P = \text{closest point from } BC \text{ to point } A;$$

$$P = \begin{cases} B, & \text{if } (x_a - x_b)\cdot(x_c - x_b) + (y_a - y_b)\cdot(y_c - y_b) < 0 \\ C, & \text{if } (x_a - x_c)\cdot(x_b - x_c) + (y_a - y_c)\cdot(y_b - y_c) < 0 \\ \left(\frac{x_a(x_c - x_b) + (y_a - a)(y_c - y_b)}{(x_c - x_b) + m(y_c - y_b)},\ a + m\cdot\frac{x_a(x_c - x_b) + (y_a - a)(y_c - y_b)}{(x_c - x_b) + m(y_c - y_b)} \right), & \text{otherwise} \end{cases}$$

The third problem, assembling the mission responsible of leading the UAV to the search path's closest point, can be solved by assembling a behavior sequence with final destination at the selected point. The straight path described by such mission can be granted as safe, for it is assumed its starting point is within a safe neighborhood of the search area, which is also assumed to be a safe collision-free area.

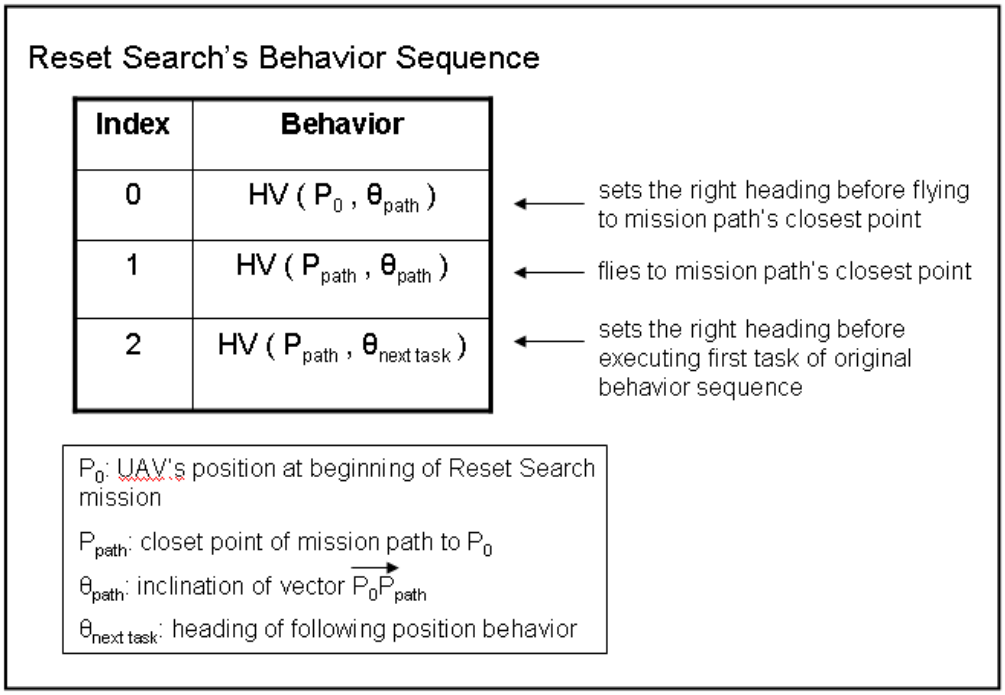

Figure 46: Reset Search mission's behavior sequence

It is also desirable follow the robustness recommendations stated in subsections **4.3.6.1** and **4.3.6.4**, which regard the use of heading parallel to UAV velocity vector and

setting the right heading before executing a Hover To task. It is also beneficial to set a Hover To task as the last task of the Reset Search mission to set the UAV with the heading of the following position behavior before restarting the Search Task in the original mission. Figure 46 illustrates the Reset Search mission's behavior sequence.

5.4 Managing the Search Object Behavior

As explained in section **4.4**, developing methods which address the problems presented by the different desirable functionalities of the Search Object behavior is not sufficient to guarantee the overall functionality of the behavior itself. It is necessary to provide the UAV's system the necessary means to apply the behavior, responding properly to external and internal events.

This section presents the module responsible for managing the different stages of a Search Object behavior. The management takes account of safety and performance efficiency of the behavior.

Forth, the following terms will be used to denote different aspects of the Search Object behavior:

- *Search Object State*: the module responsible for managing the events which could take place during a Search Object behavior and the different stages on the temporal evolution of the behavior.
- *Search Object Behavior Object*: the module responsible for realizing the functionalities required by the Search Object behavior. This denomination is inspired by the implementation of the behavior using Object Oriented Programming.

5.4.1 Search Object Behavior Modeling

The Search Object behavior was modeled according to UML specifications for Statechart diagrams. This choice was based on the advantages of statechart-based modeling as presented on subsection **3.1.1**. This model was conceived with the purpose of managing possible events during a mission managed by the Search Object behavior, as well as allowing the UAV to execute different stages of the behavior.

As stated in section **3.1**, the Search Object state is part of the state-chart diagram designed to model the Supervisor module. The system transits to Search Object state when the current mission executed by ARTIS starts a Search Task which defines a valid search area perimeter, under the condition that the operator has granted permission to have Search Tasks managed by Search Object behavior. Figure 47 presents the State Chart model designed for the Search Object behavior.

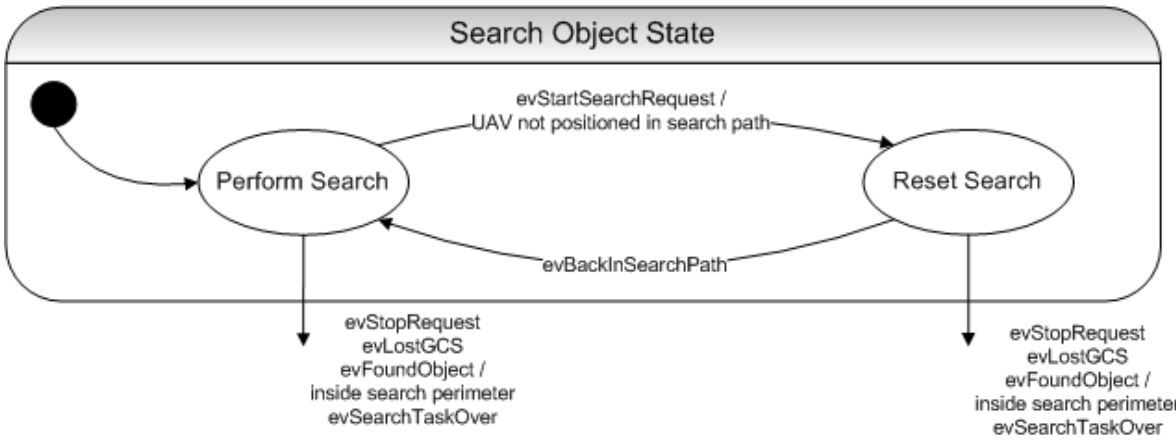

Figure 47: Search Object state chart model

The Search Object state comprises two sub-states, namely *Perform Search* and *Reset Search*. The default sub-state of Search Object state is Perform Search sub-state. Bellow follows a brief description of each sub-state.

- *Perform Search:* the system transits to this sub-state at the beginning of the Search Object behavior. If, at moment when Perform Search sub-state is entered, the UAV is not following the search path, the system transits to sub-state Reset Search. While the system performs a regular Search Task, it remains in this state.

- *Reset Search:* the system transits into this sub-state if it is necessary to return to the search path before recommencing the Search Task. In the state chart above, this event is denoted by *evStartSearchRequest / UAV not positioned in search path*. The system remains in this sub-state while the UAV has not reached a point of the search path.

The two sub-states presented are described in more detail in subsections **5.4.1.1** and **5.4.1.2**.

Below follows the description of the events relevant during while the system is in Fly Home state.

- *evStopRequest*: occurs when GCS issues a Stop command to the UAV;

- *evLostGCS*: occurs when the UAV's embedded system looses contact with GCS and the operator;
- *evFoundObject*: occurs when the system's vision computer (VC) signalized the recognition of the searched ground object. The guard condition *inside search perimeter* is true when both the UAV's and ground object's estimated positions are within the search area perimeter.
- *evSearchTaskOver*: occurs when all of the tasks described by the behaviors in the relative Search Task have been successfully performed to completion.
- *evStartSearchRequest*: occurs when the system transits to Search Object state. The guard condition *UAV not positioned in search path* is true if the current mission does not indicate the immediate start of a behavior of a Search Task of the mission. In other words, the UAV is not positioned in the planned search path.
- *evBackInSearchPath*: occurs when the UAV's position is on a point belonging to the search path.

5.4.1.1 Perform Search

In the Perform Search sub-state, the Search Object Behavior object uses the behavior sequence describing the original mission to compute the perimeter of the search area if it has not been calculated in a previous access of the state. The computation of the search area perimeter is described in subsections **5.3.1** and **5.3.2**. Also when entering the sub-state, the mission is stored in the Search Object Behavior object, for it might be necessary to reset the mission in other stages of the behavior.

The flowchart in figure 48 illustrates the processes and features executed by the Search Object Behavior object under responsibility of Perform Search sub-state.

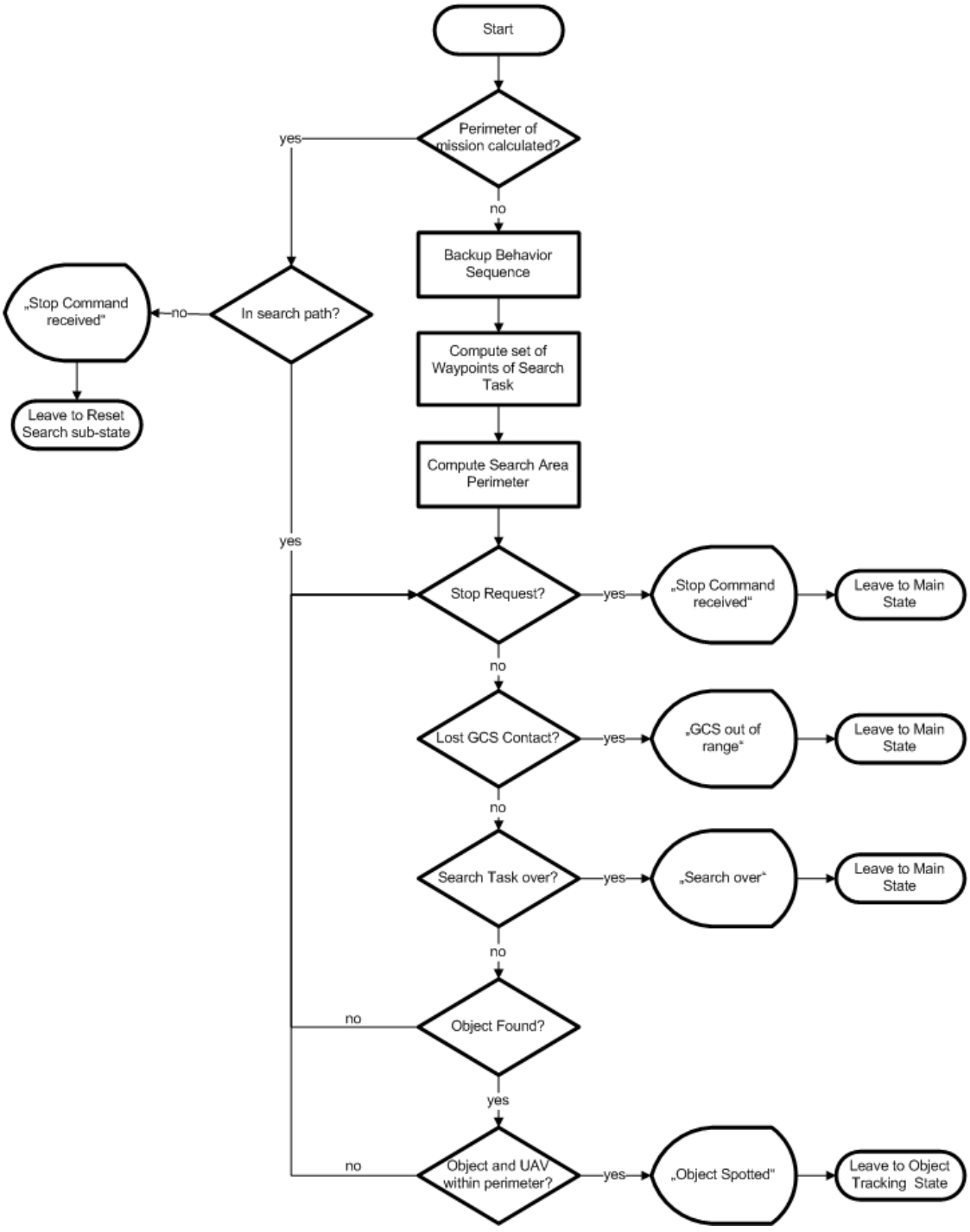

Figure 48: Flowchart description of Perform Search sub-state

5.4.1.2 Reset Search

In the Restart Search sub-state, the Search Object Behavior object is in charge of calculating which point of the search path is the closest to the position of the UAV at the moment the sub-state is entered. Furthermore, the Search Object Behavior object is in charge

of computing the mission which leads the UAV back to the search path. This mission is computed according to the procedures described in subsection **5.3.3**. This mission is forth referred to as *Reset Search mission*. Figure 49 illustrates a Reset Search mission.

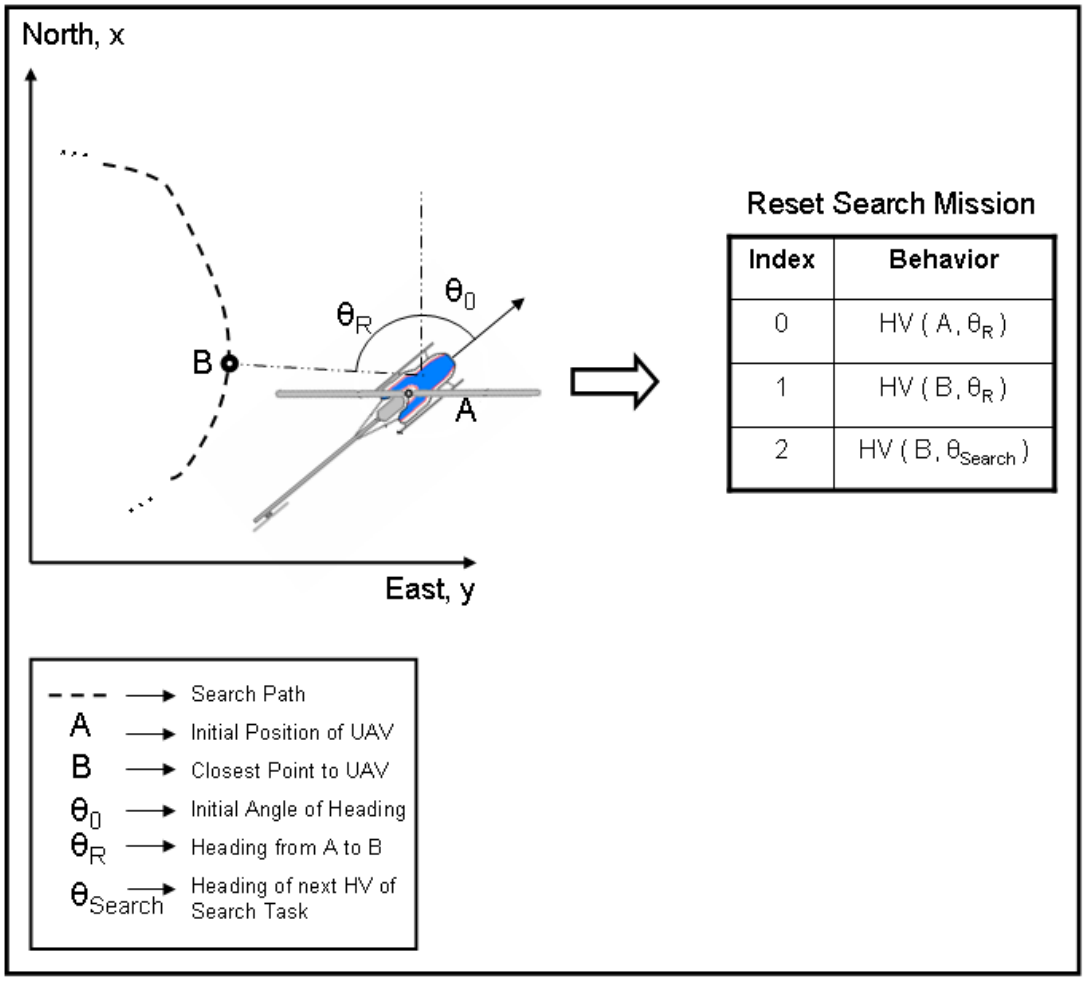

Figure 49: Reset Search Mission

Another attribute of the Reset Search sub-state is running the Reset Search mission and monitoring the events during the its course. Once the mission is completed and the UAV has reached the search path, the Search Object Behavior object must reset the old mission with index corresponding to the segment of search path where the helicopter is. If the object is spotted before the completion of the Reset Search mission, the sub-state is in charge of realizing the proper transition to Object Tracking state.

The flowchart in figure 50 illustrates the processes and features executed by the Search Object Behavior object under responsibility of Reset Search sub-state.

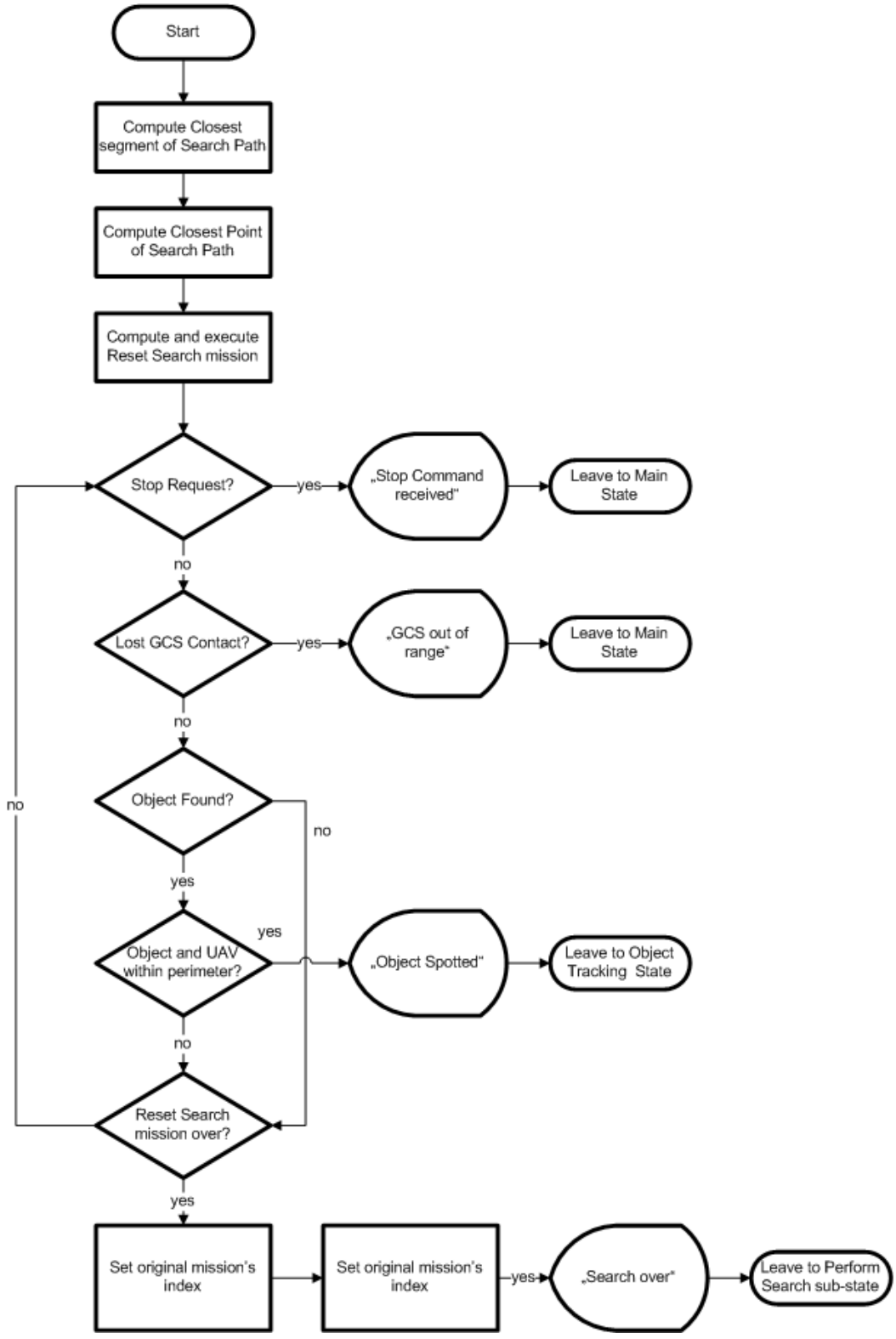

Figure 50: Reset Search flowchart representation

5.5 Results

This section presents the practical results obtained with the implementation of the Search Object behavior, as well as the analysis of such outcome. One of the features tested in this section is the calculation of the search area cells' perimeters. The feature concerning resetting (i.e. re-starting the original search mission at an appropriate stage) the search mission after the system has finished an object tracking task is also tested, along with the management capabilities of the behavior.

It is important to remember that, unless explicitly remarked otherwise, the coordinates shown in this section are conformant to the NED reference system, i.e. (North [m], East [m], Down [m]) set at ground level.

5.5.1 Search Area

In this subsection we present the tests made to verify the Search Object behavior's capability of recognizing and calculating the set of convex cells which define the search area. The scenario proposed was a searched mission around the airport of Hannover planned by an operator. Figure 51 shows the planning.

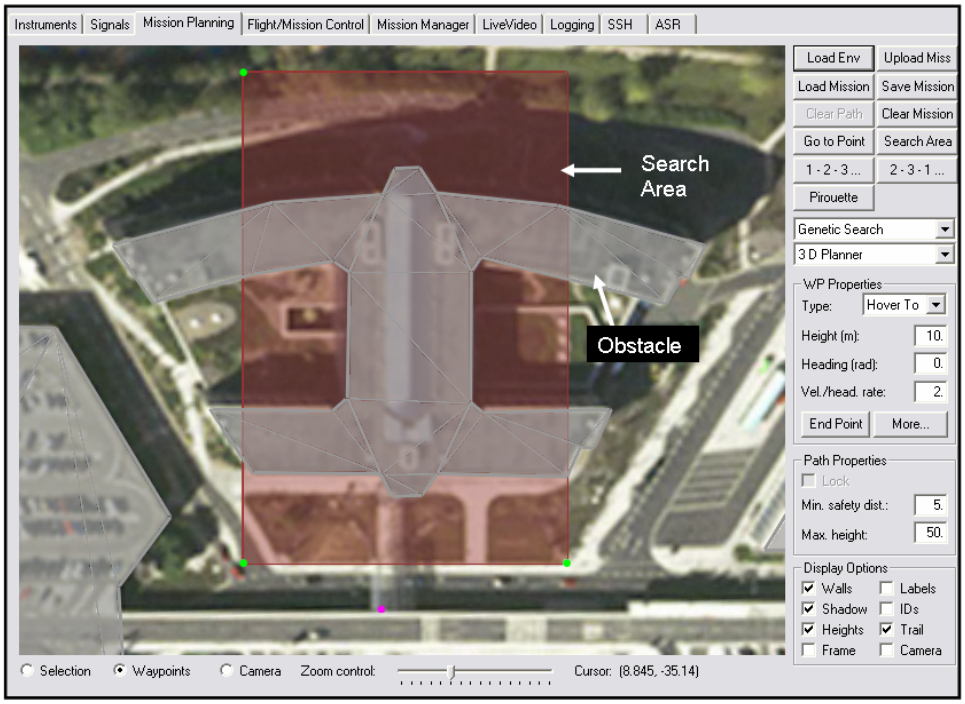

Figure 51: Test's search area

The search area presented in figure 51 was set by the operator and represents the area in which the operator desires to conduct the search for the ground object. Within such area there is a building, which constitutes an obstacle. Figure 52 presents the path generated by the GCS intended to cover the area.

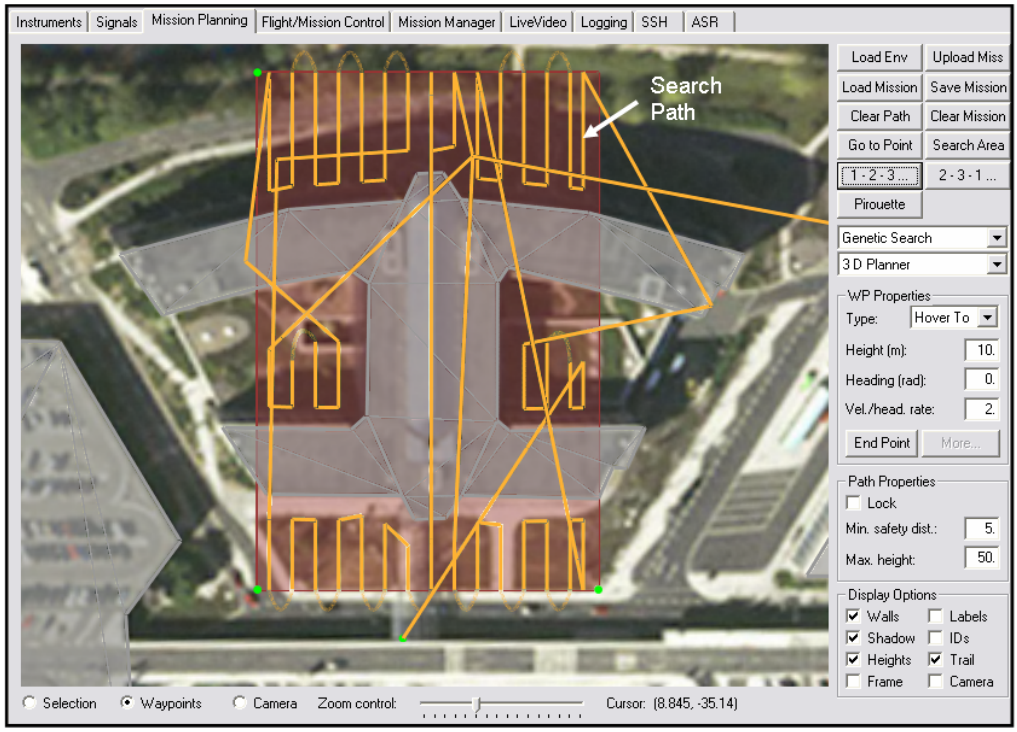

Figure 52: Test's search area and search path

As shown in figure 52, the planner divides the total search area into obstacle free cells with optimized arrangement, i.e. area sectors contained by the total search area, which are within a safe distance from the obstacle. The Search Object behavior must thus be able to recognize and compute the boundaries of each cell. Furthermore, the behavior must be able to distinguish the path segments destined to cover each search cell and the segments destined to lead the UAV from a search cell to another, e.g. the segments of path flying over the obstacle. The related mission was used by the behavior to compute the cells as shown in figure 53.

As shown in figure 53, the behavior successfully calculated the convex cells of the total search area based on the waypoints of the search mission. The results were confirmed using Unit testing. Figure 54 illustrates the computed search convex cells in the original plan.

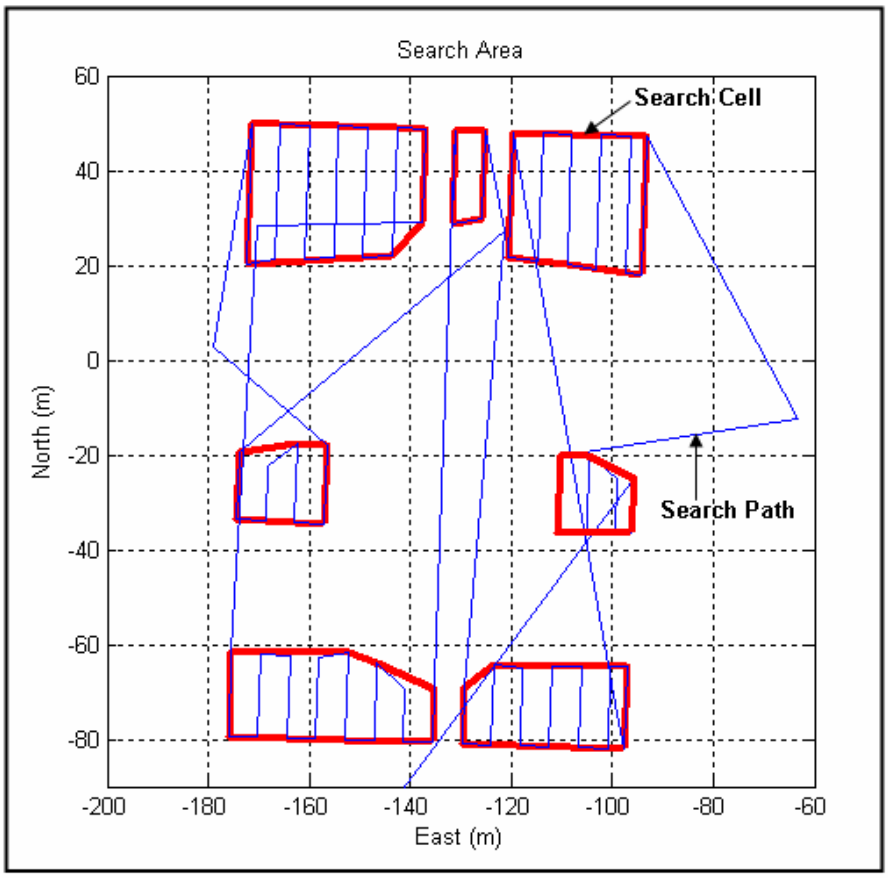

Figure 53: Search convex cells computed from search mission

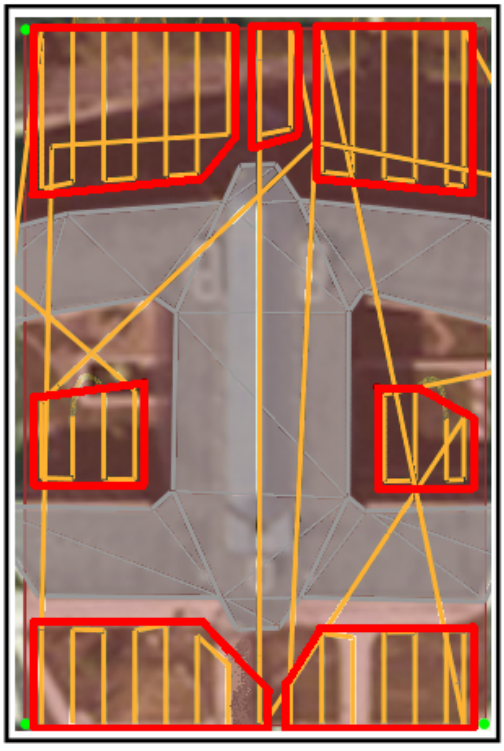

Figure 54: Search convex cells in the original search mission

5.5.2 Resetting a Search Mission and Behavior Management

When the Search Object behavior must reset a mission, one of the steps is assembling a mission leading the UAV back to the closest path segment of the search path and repositioning the UAV's heading properly before re-initiating the original mission. The first test presented in this subsection concerns such feature.

Using Unit testing, a search mission was planned without assistance of MAESTRO. Three scenarios in which the Search Object behavior must reset a search mission were proposed, as presented on figure 55. In each scenario, the UAV's is in a certain position out of the search path, and we tested if in each scenario the mission planned by the behavior lead properly back to the search path.

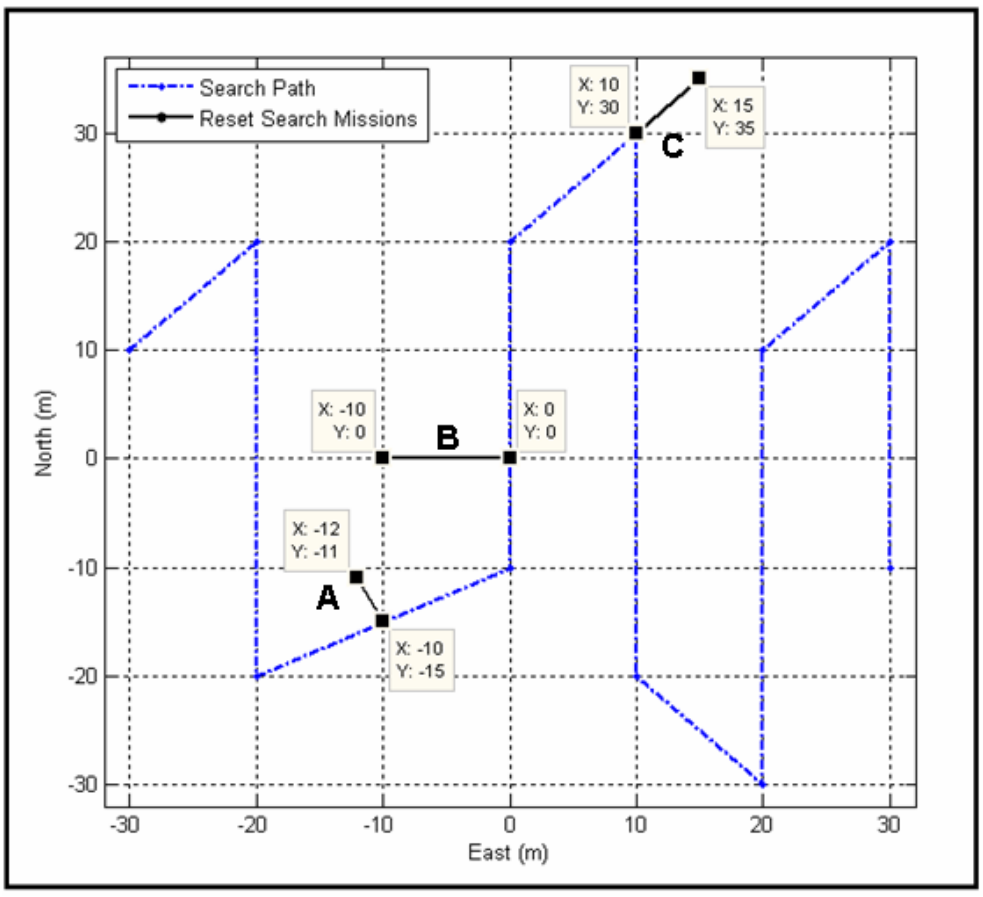

Figure 55: Reset mission unit test

In scenario **A**, the helicopter's position before the mission resetting was at coordinates (-11, -12, -9) [m]. In the test, the Search Object behavior provided a provisory mission leading back to the search path at point (-15, -10, -9) [m].

In scenario **B**, the position before the resetting was at coordinates (0, -10, -9) [m]. In such scenario, the behavior provided a mission leading back to the search path at coordinates (-15, -10, -9) [m].

In scenario **C**, the helicopter's position before the mission resetting was at coordinates (35, 15, -9) [m]. In the test, the behavior provided a mission leading back to the search path at point (30, 10, -9) [m].

In each scenario, it was validated through the Unit test that the mission provided by the Search Object behavior lead back to the search track properly while respecting the heading recommendations (subsections **4.3.7.1**, **4.3.7.4**).

The second test concerning the resetting search feature was realized with the use of SITL simulation. The objective was to assure that a reset search mission could be properly assembled and executed *online*. In this test, the search mission shown in figure 56 was planned offline using MAESTRO.

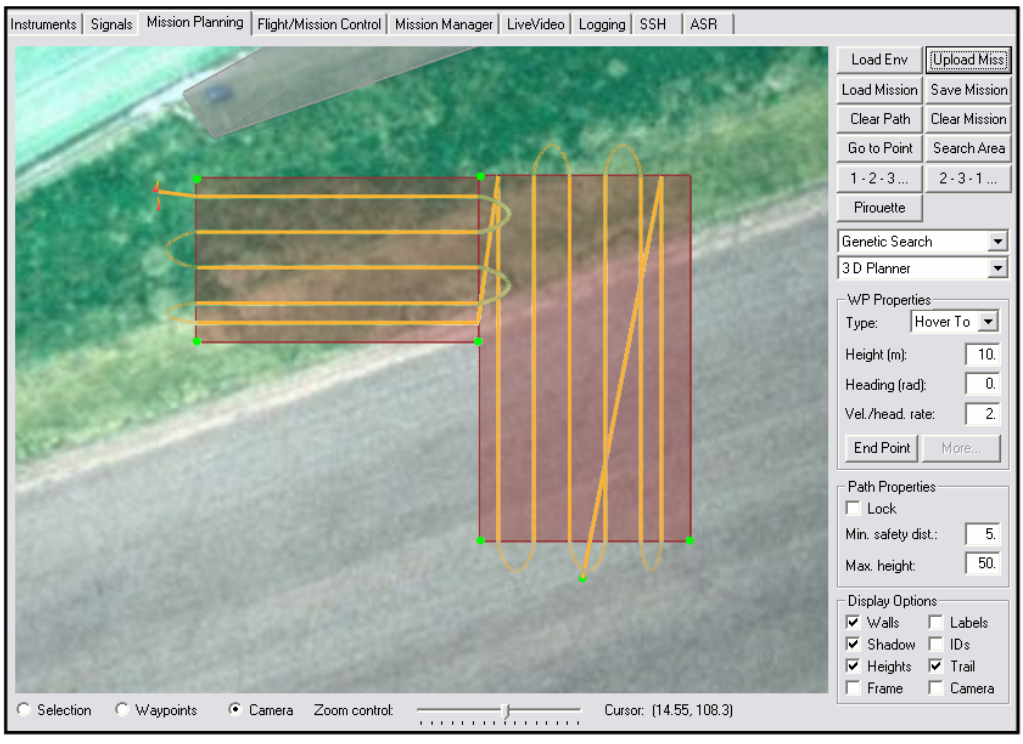

Figure 56: Reset mission SITL test, mission plan

The mission constitutes in a concave search area divided into two convex cells. The test was intended to check if the Search Object behavior was capable of properly restarting a search mission after the system finished performing a track object task. Figure 57 shows the path of the aircraft in such simulation.

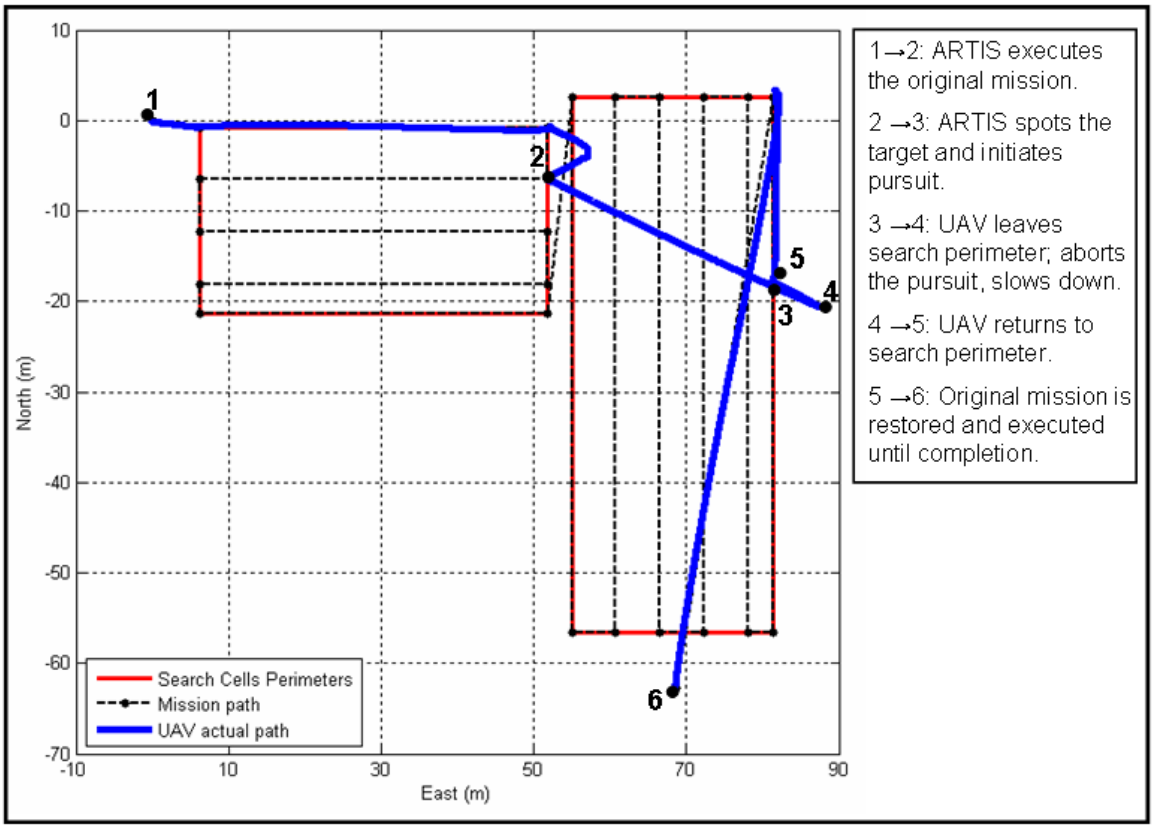

Figure 57: Reset mission SITL test, execution

In figure 57, the system executes the original mission from point **1** to point **2** without interference of intelligent behaviors. After spotting the target on point **2**, the Object Tracking behavior was activated and conducted the pursuit of the object until point **3**, when the aircraft left the search area. From point **3** till point **4**, the behavior commanded the UAV to slow down, and then to return to the search area's border at point **5**. At this point, the Search Object behavior was activated.

After activation, the Search Object behavior recognized that its position was at the search path. Therefore, the behavior analyzed the mission and determined which behavior would be responsible for the path segment in which the UAV currently is. After setting the heading pointed to the waypoint defined by such behavior, Search Object properly restarted the execution of the original mission in the correct stage and finished the original mission when reaching point **6**.

In this scenario, the Search Object behavior was also tested with regard to its management capabilities, since the test was realized online and the state transitions and processes must have been executed properly in order to secure the success of the test. Figure 58 presents the temporal evolution of the test.

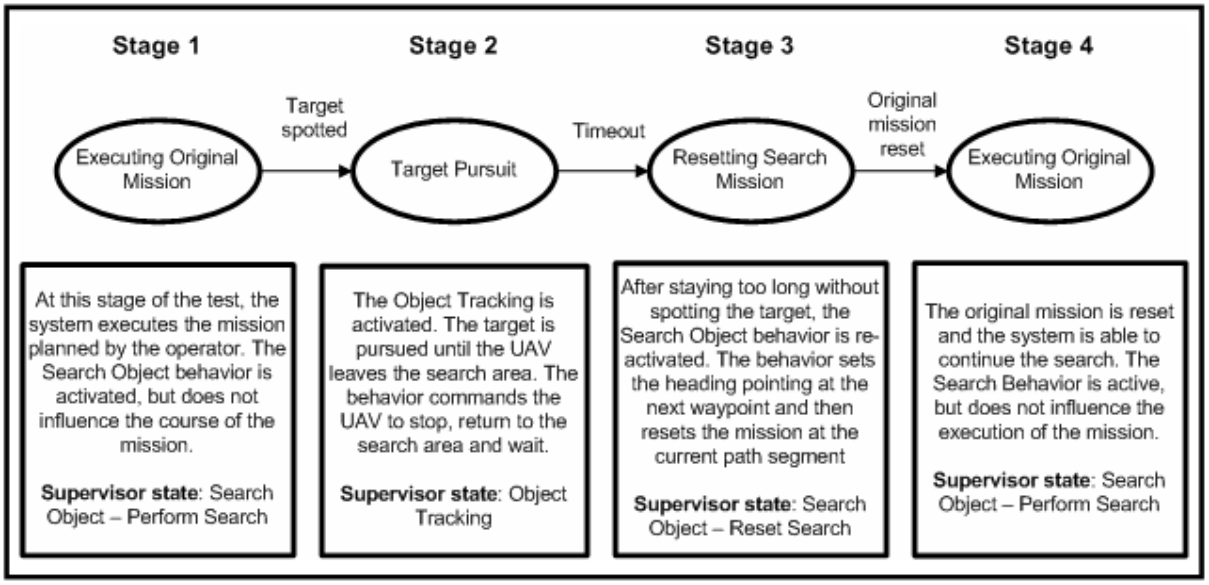

Figure 58: Temporal evolution of a reset mission SITL test

The presented temporal evolution of the reset search task was checked using SITL simulation and corresponds to the expected functionality of the behavior, as shown on section **5.4**.

6 Object Tracking

This chapter describes the intelligent behavior module designated *Object Tracking*. This module has been conceived, implemented and successfully tested. This behavior provides ARTIS with the capability of pursuing a moving ground object.

The ARTIS project had already developed a module capable of commanding the UAV to pursue a ground object [27]. However, this module would simply command the aircraft to hover to the position determined by the VC. Such approach disregarded aspects such as the object's motion dynamics, camera noise and possible obscuring of the object. Also, once the object was lost the tracker would not "know" where to look at, e.g. if the object was lost while it was moving north, the module would not take the intuitive action of also commanding northwards motion. Furthermore, the module had no "awareness" of its function in the context of the whole mission, i.e. the module would completely disregard the original mission when performing the tracking.

The objective if the Object Tracking behavior is, therefore, addressing aspects ignored by the old tracker, enhance the performance of the tracking and providing ARTIS with the capability of acting "intelligently" in the target pursuit, taking intuitive actions during the tracking, such as flying higher to enhance camera view when the object is lost and avoiding pursuing the target in areas outside the search area.

6.1 Problem Statement

The Object Tracking intelligent behavior addresses the problem of controlling the UAV's movement in order to maintain visual contact with a mobile ground object and managing an Object Tracking mission. These two aspects of the problem are analyzed independently. Forth, we will refer to the two aspects of the problem as the *Tracking sub-problem* and the *Management sub-problem*.

The data regarding the position of the ground object is provided to the control system by ARTIS' vision computer (VC). The VC makes use of data acquired from a monochrome camera pointing downwards, providing a ground area view as demonstrated by the following figure. A pattern recognition program enables the VC to perceive the presence of the ground

object and estimate its position relative to the UAV [27]. Therefore, the problem of visually recognizing the ground object is not addressed by this behavior module.

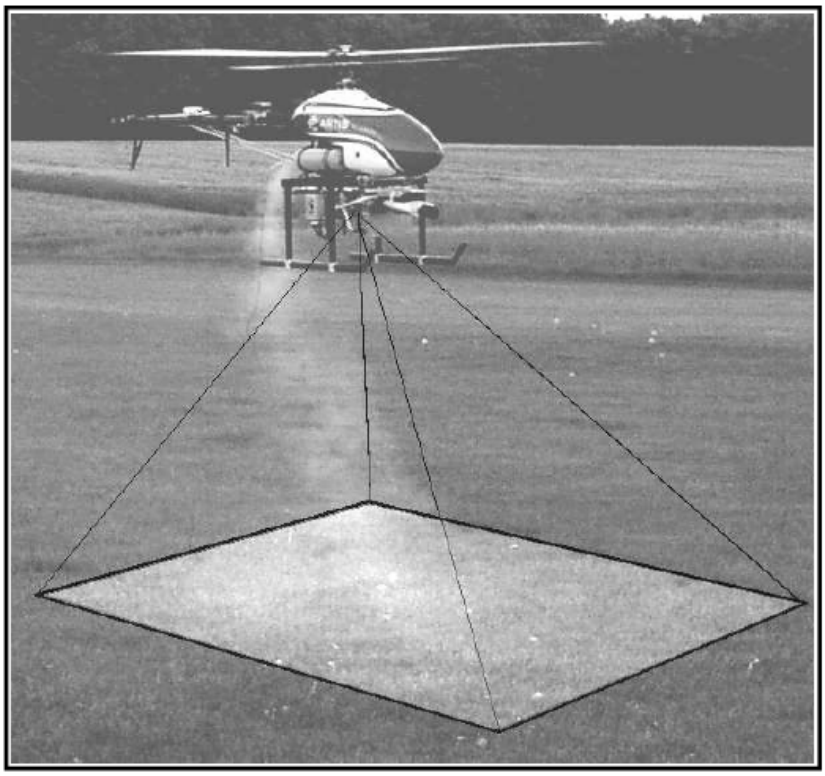

Figure 59: Ground view of ARTIS' camera

The Tracking sub-problem comprehends the problems of interpreting the data provided by the VC, estimating the position of the ground object based on this data and developing efficient movement strategies to maintain the ground object within the range of the ground camera. The nature of the movement of the ground object is assumed to be unknown. Moreover, the environment within the search perimeter (subsection **5.3.1**) is assumed to be safe and collision-free.

The Management sub-problem comprises the problems of providing safe and efficient transitions to and from the Object Tracking state (e.g. when transiting from Search Object state to Object Tracking state), managing the internal transitions of the Object Tracking state (e.g. when the VC provided no visual data for an extended time period) and recognizing situations which signify a failure of the Object Tracking mission (e.g. when the object or the UAV evade the area in which the UAV is allowed to fly by the operator).

6.2 Discussion

As previously mentioned, before the implementation of the Object Tracking behavior, the ARTIS project already had the capability of pursuing ground objects visually detected by the on-board camera. The approach used was to use the data of the VC regarding the ground target and use ARTIS' waypoint navigation system to command the UAV to the northern and eastern coordinates of such position while maintaining constant height. However, the efficiency of the pursuit was not satisfactory. The camera's information on the target's position is not accurate, and therefore the control commands do not meet the objective of hovering to the object's coordinates. Furthermore, the response to the target's locomotion is not quick, for the approach ignores the target's motion dynamics. Moreover, the system's control system is such that the UAV moves slowly to positions close itself. All these aspects of the previous target tracker contribute for undesirable effects of a tracking system that can be easily deceived by the object's motion and often looses track of the target for responding too slow. The approach proposed for the Object Tracking behavior is intended to address and improve all the negative aspects of the past target tracker.

Another aspect of the past object tracking system which presented room for improvement was the fact that it did not consider the context of the mission, responding simply and unconditionally by moving towards the object. Situations such as leaving the search area are not regarded. Moreover, loosing visual contact with the target causes the system to stand by, waiting for further instructions from the operator. Increasing the on-board system's awareness concerning the agent's mission and providing it with the capability to properly and autonomously handle events such as the ones mentioned without necessity of intervention from the operator increases the system's overall autonomy [22], [23]. Therefore, the proposed approach for the design of the Object Tracking behavior addresses such issues.

As stated in the previous section, the problem addressed by the Object Tracking behavior can be decomposed into several specific problems. The following subsection discusses the nature of these problems, reviews, when pertinent, related works documented in the literature and establishes the approaches of this work concerning these problems.

6.2.1 Target Position Estimation

The problem of estimating the position of an object based on noisy sensors has been addressed in a variety of studies and applications, ranging from radar target tracking [38] to tracking of groups of targets [39]. In the problem addressed by the Object Tracking behavior, no assumptions are made regarding the ground object's motion characteristics. It is assumed that the ground object tracked is unique, excluding the possibility of multiple targets.

Bayes filters are widely used in the estimation an object's location. The most widely used variant of Bayes filters is the Kalman filter [40], a set of mathematical equations that provide a recursive method to estimate the state of a process in a way that minimizes the mean of the squared error [41]. The main advantage of the Kalman filter consists in its computational efficiency, and its performance is best when the uncertainty in the state is not too high. The Kalman filter has been applied with great success to several tracking problems [40].

Another example of Bayes filter used as a location estimator is the Particle Filter [40], whose idea is to apply a recursive Bayesian filter based on sample sets [42]. The main advantage of the Particle Filter is their ability to represent arbitrary probability densities [40]. One disadvantage of the method is that worst-case complexity grows exponentially in the dimensions of the state space [40].

Some works applied Interacting Multiple Model (IMM) estimators in the tracking problem [38], [39]. Such estimator is a suboptimal hybrid filter whose main feature is the ability to estimate the state of a dynamic system with several modes which can "switch" from one to another [43]. The IMM algorithm has shown good performance when tracking targets with high maneuverability [44].

In comparison with the Particle filter, the Kalman filter has the advantage of having higher operation efficiency [40]. However, the Particle filter has higher robustness when compared to the Kalman filter, since it has the ability to represent arbitrary probability densities [40]. The IMM approach also shows advantages when compared to previous filters because it gets around the difficulty due to the model uncertainty by using more than one model [45]. The disadvantage of the IMM relies in the fact that it decides which model to apply before the estimation ("decision followed by estimation"), and therefore possible errors in the decision on the model are not accounted for in the estimation [45]. Moreover, implementing several models, and consequently several filters, requires a more thorough implementation effort than implementing a single filter.

Because of its well documented success in tracking application, its computational efficiency and implementation simplicity, the Kalman filter was the chosen approach for the estimation of the ground object's position. The IMM estimator was also considered, for its multi-model approach addresses the issue of the uncertainty in the target's motion model. However, time restraints impeded its more sophisticated implementation.

6.2.2 Pursuit Strategy

The problem of developing strategies to pursue a moving object has been widely addressed through literature.

Such problem can be related to *Pursuit – Evasion Contests* game theory [46], a family of mathematical problems in which one group attempts to track down members of another group in an environment[†]. Such theory has been applied to the coordination of a team of UAVs and Unmanned Ground Vehicles (UGV) when pursuing a group of evaders [47].

Other authors address the related problem of pursuing a target in different scenarios and with distinct objectives. Some authors addressed the problem of maintaining a moving target within sensing range of an observer reacting with delay [48]. In this work, the region around pursued and pursuer is divided in regions in which the target could escape from sensing range and regions in which it could not escape. The motion strategy proposed involves keeping the segment connecting pursuer and pursued in non-escape regions.

Some authors addressed the problem of tracking a ground object with a UAV with the approach of predicting the future trajectory of the target using Artificial Neural Networks (ANNs) [49]. In such work, the observation of the behavior pattern of the target allows the ANNs to learn its dynamics in the process, assisting in the prediction of the objects trajectory. This problem has also been addressed by other authors with the estimation of the target's position and the use of a specially-tuned waypoint navigator [50].

The approach proposed in this work to address the problem of pursuing a ground target consists in using predictions of the target's future position and using a modification of ARTIS' waypoint navigator to command the UAV to hover to such position. By considering the object's motion dynamics, the response to its locomotion is more adequate. Furthermore, the future position of the object is further away from the UAV than the present position when the target moves away from the agent. A position command to a point further away implies in

[†] http://en.wikipedia.org/wiki/Pursuit-evasion

more vigorous control signals, enhancing the action speed and enhancing the system's capability of keeping the target within camera view range. By introducing a method to predict the object's future position, we also present the means for the UAV to keep flying to the positions which the behavior *expects* to find the target.

6.2.3 Other Issues

One feature designed for the Object Tracking behavior was the capability to analyze the UAV's position with respect to the search area of the Search and Track mission (if one is defined) and to indicate if the aircraft is within the boundaries of the search area. As explained on subsection **5.2.3**, the search area is divided into convex cells. The approach used to implement such feature was checking if the UAV is positioned within each of the convex cells or nearby two or more cells, which indicates that the helicopter is located within two convex cells of the same concave area.

The problem of determining if the UAV is within a convex cell can be related to the *Point in Polygon* computational geometry problem, which aims to check if a given point in the plane lies inside, outside, or on the boundary of a polygon[†]. The approach used to solve this problem was applying the *Jordan Curve Theorem*[‡] to check if the point defined by the northern an eastern coordinates of the UAV's position is within the polygon defined by the convex cell's perimeter.

As discussed for the Fly Home and Search Object behaviors, it is necessary to provide the Object Tracking behavior with the capability to manage the Tracking task, providing proper responses to events and managing the transition to different stages of the task. Situations addressed by the behavior's management capabilities include the fled of the UAV or target form the search area, the first spotting of the target and the loss of sight of the target for an extended period. As implemented for the other intelligent behaviors, the approach chosen to provide the Object Tracking behavior's management capabilities was modeling the behavior as using UML's statecharts.

[†] http://en.wikipedia.org/wiki/Point_in_polygon
[‡] http://en.wikipedia.org/wiki/Jordan_curve_theorem

6.3 Tracking Sub-problem

This section presents the analysis of possible solutions and the development of the solutions proposed for the two subjects of the Tracking sub-problem: the estimation of the ground object's position based on data provided by the VC and the target pursuit movement strategy.

6.3.1 Estimation and Prediction of the Target's Position

As stated earlier on this chapter, the approach proposed to address the problem of estimating the ground object's position was using a Kalman filter. The usage of the Kalman filter is useful to estimate future positions of the target, as well as providing estimates its position when there's lack of visual data from the VC. This way, the system is capable to predict the object's position and continue the tracking even during moments when the object is obscured or out of range. This capability is fundamental, since the aircraft often looses visual contact with the ground object during critical situations, such as when it suddenly changes the direction of movement, as shown in figure 60.

Another aspect of the system which favors the filtering of the VC data is the fact that such data is subject to measurement noise. Figure 61 presents the data from the VC collected in a HITL simulation.

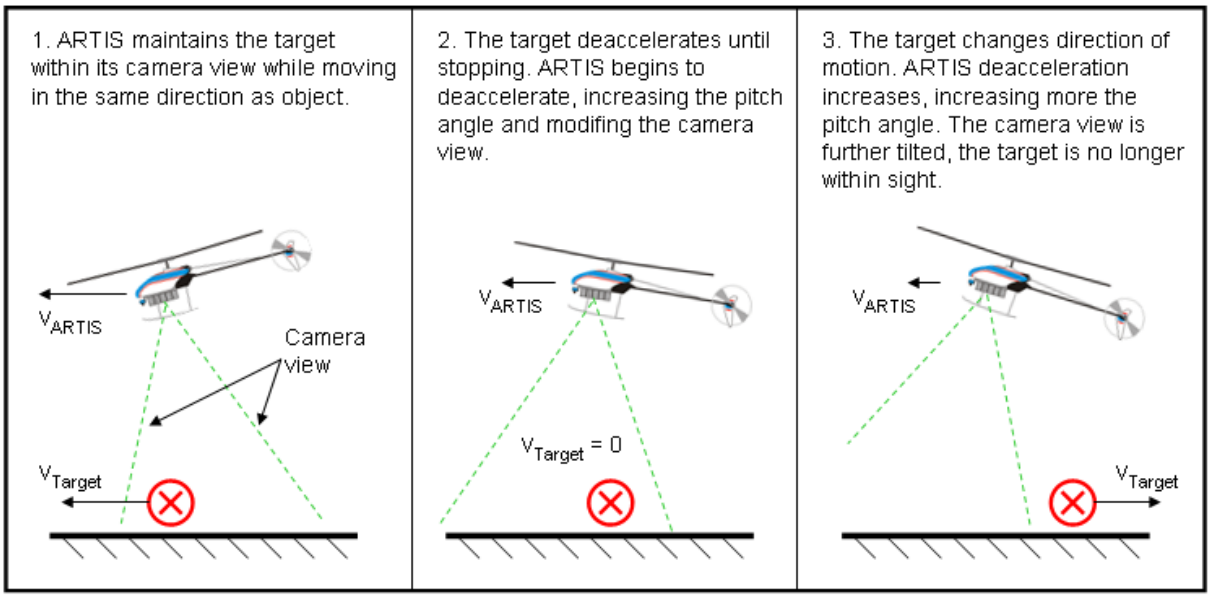

Figure 60: ARTIS' motion's influence on camera view

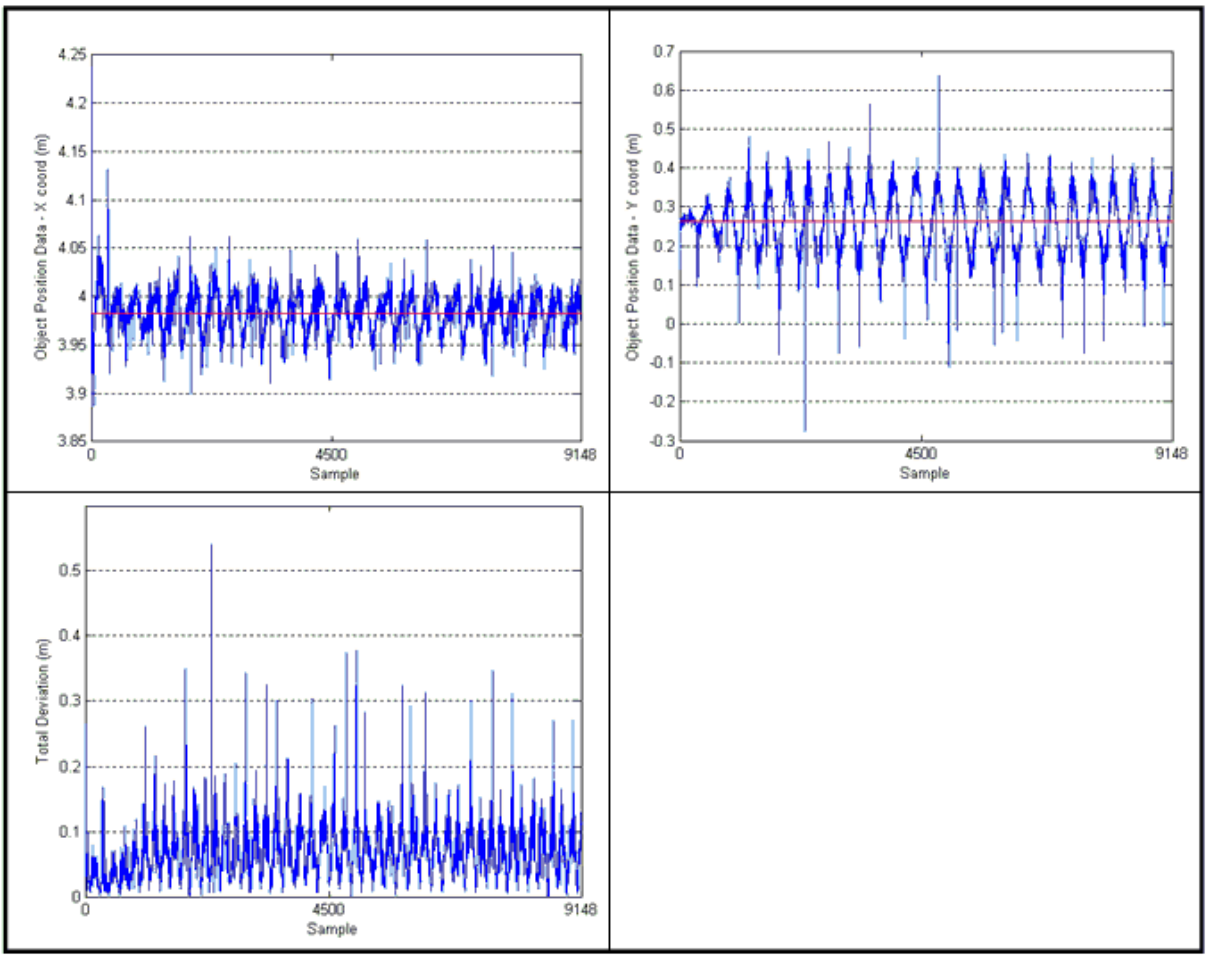

Figure 61: Camera measurement noise

In the simulation, the UAV was set to hover around the ground object in an altitude of 8 meters, however keeping the object within sight. The object remained motionless, and its horizontal and vertical coordinates are assumed to be the average of the measurements, shown in the picture. As the camera produces noisy position measurements, the use of a Kalman filter is useful to estimate the real position of the object.

As the target motion's characteristics are assumed to be unknown, five Kalman filters with different process models were developed and the respective performances were evaluated. Before presenting the filters, we present a brief explanation about the functioning of the discrete Kalman filter.

6.3.1.1 The Discrete Kalman Filter

The Kalman filter is a recursive method to the discrete linear problem of estimating the true state of a certain process[†]. The process to be estimated is represented as follows in (6-1) and (6-2):

$$x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1} \tag{6-1}$$

$$z_k = Hx_k + v_k \tag{6-2}$$

In (6-1), the matrix A (n x n) (assumed to be constant) relates the current state x_k (n x 1) with the state in the previous step $k - 1$. The matrix B (n x m) relates the optional control input u (m x 1) to the state x. The matrix H (p x n) in equation (6-2) relates the state x_k with the measurement z_k (p x 1). The random variables w_k and v_k represent, respectively, the process and measurement noise. They are assumed to be mutually independent, white and with normal probability distributions as follows in (6-3) and (6-4):

$$p(w) \sim N(0, Q) \tag{6-3}$$

$$p(v) \sim N(0, R) \tag{6-4}$$

In such conditions, it can be proved that the Kalman filter algorithm produces a state estimation which minimizes the mean of the squared error [41]. The algorithm is described as follows.

The algorithm is divided into two stages, a *prediction* stage and a *correction* stage. In the prediction stage, the state in step k is estimated using a prediction based on the estimated state in step $k-1$, resulting in the *a priori* estimation $\hat{x}_k^-$ (n x 1). In the correction stage, the prediction and the measurement at step k are used to generate an *a posteriori* estimate $\hat{x}_k$ (n x 1). Let the *a priori* and *a posteriori* estimate errors be represented as:

$$e_k^- \equiv x_k - \hat{x}_k^- \tag{6-5}$$

$$e_k \equiv x_k - \hat{x}_k \tag{6-6}$$

The covariance of the *a priori* and *a posteriori* estimate errors are thus represented as:

$$P_k^- = E[e_k^- e_k^{-T}] \tag{6-7}$$

$$P_k = E[e_k e_k^T] \tag{6-8}$$

[†] The explanation about the Kalman filter is based on the description on [32]. For a more detailed description of the algorithm, consult the reference.

The prediction and correction stages are executed through the equations in the tables 4 and 5.

Table 4: Discrete Kalman filter prediction equations

$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_{k-1}$	(6-9)
$P_k^- = AP_{k-1}A^T + Q$	(6-10)

Table 5: Discrete Kalman filter correction equations

$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1}$	(6-11)
$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-)$	(6-12)
$P_k = (I - K_k H)P_k^-$	(6-13)

These are the steps necessary for the implementation of the discrete Kalman filter. The process estimated in the problem of estimating the object's position is unknown. Therefore, each of the five filters has been implemented with a different process model (6-1) and a respective measurement process (6-2).

Table 6 presents symbols used in the following subsections to represent the entities involved.

Table 6: symbols and representations

Symbol	Entity
r_k^i	Target's position [m] in cycle *k* and direction *i*
v_k^i	Target's velocity [m/s] in cycle *k* and direction *i*
a_k^i	Target's acceleration [m/s^2] in cycle *k* and direction *i*
da_k^i	Target's acceleration derivative [m/s^3] in cycle *k* and direction *i*
$z_k^{r,i}$	Target's position measurement [m] in cycle *k* and direction *i*
T	Period of each cycle [s]

6.3.1.2 Filter 1

The first filter developed used the model presented in table 7.

Table 7: Kalman filter 1

$$\begin{bmatrix} r_k^x \\ r_k^y \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{k-1}^x \\ r_{k-1}^y \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v_{k-1}^x \cdot T \\ v_{k-1}^y \cdot T \end{bmatrix} + w_{k-1}^r \quad (6\text{-}14)$$

$$\begin{bmatrix} z_k^{r,x} \\ z_k^{r,y} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_k^x \\ r_k^y \end{bmatrix} + v_k^r \quad (6\text{-}15)$$

$$v_{k-1}^i = \frac{r_{k-1}^i - r_{k-2}^i}{T} \quad (6\text{-}16)$$

In this filter, the position coordinates are considered to be independent. The velocities in the x and y direction in cycle *k-1* are calculated using the estimated positions on cycles *k-1* and *k-2*, and influence the position prediction as if they were *inputs* of the system.

6.3.1.3 Filter 2

The idea of the second filter is using two independent Kalman filters to estimate the position (6-18 and 6-19) and the velocity of the object (6-20, 6-21 and 6-22). The estimation provided by the velocity filter is intended to improve the position filter's performance. The second filter was developed using the model described in table 8.

In this filter, the position coordinates as well as the velocity components are considered to be independent. The accelerations in the x and y direction in cycle *k-1* are calculated using the estimated velocities on cycles *k-1* and *k-2*, and influence the position and velocity predictions as if they were *inputs* of the system.

The filter functions as follows: after a measurement, the position vector in cycle *k* is estimated using the velocity and acceleration vector as inputs in (6-18). The measurement of the velocity vector is then described by equation (6-17).

$$\overrightarrow{v_k} = \frac{\overrightarrow{r_k} - \overrightarrow{r_{k-1}}}{T} \quad (6\text{-}17)$$

Both position vectors in (6-17) are estimated positions. After such measurement, the velocity vector is estimated considering the acceleration vector as an input in (6-20).

Table 8: Kalman filter 2

$$\begin{bmatrix} r_k^x \\ r_k^y \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{k-1}^x \\ r_{k-1}^y \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v_{k-1}^x \cdot T + a_{k-1}^x \cdot \frac{T^2}{2} \\ v_{k-1}^y \cdot T + a_{k-1}^y \cdot \frac{T^2}{2} \end{bmatrix} + w_{k-1}^r \quad (6\text{-}18)$$

$$\begin{bmatrix} z_k^{r,x} \\ z_k^{r,y} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_k^x \\ r_k^y \end{bmatrix} + v_k^r \quad (6\text{-}19)$$

$$\begin{bmatrix} v_k^x \\ v_k^y \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v_{k-1}^x \\ v_{k-1}^y \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} a_{k-1}^x \cdot T \\ a_{k-1}^y \cdot T \end{bmatrix} + w_{k-1}^v \quad (6\text{-}20)$$

$$\begin{bmatrix} z_k^{v,x} \\ z_k^{v,y} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v_k^x \\ v_k^y \end{bmatrix} + v_k^v \quad (6\text{-}21)$$

$$a_{k-1}^i = \frac{v_{k-1}^i - v_{k-2}^i}{T} \quad (6\text{-}22)$$

6.3.1.4 Filter 3

The idea of the third filter is using three independent Kalman filters to estimate the position (6-23 and 6-24), velocity (6-25 and 6-26) and acceleration (6-27, 6-28 and 6-29) of the object. The estimations provided by the velocity and acceleration filter are intended to improve the position filter's performance. The third filter was developed using the model presented in table 9.

In this filter, the position coordinates as well as the velocity and acceleration components are considered to be independent. The acceleration derivative vector in cycle *k-1* is calculated using the estimated acceleration on cycles *k-1* and *k-2*, and influences the position, velocity and acceleration predictions as if it was an *input* of the system.

The filter functions in a similar fashion of the previous filter. After a position measurement, the position vector in cycle *k* is estimated using the velocity, acceleration and acceleration derivative vector as inputs in (6-23). The measurement of the velocity vector follows equation (6-22). After measuring and estimating the position vector, the velocity vector is measured and estimated using the acceleration and acceleration derivative vectors as inputs in (6-25). The measurement of the acceleration is described by equation (6-30).

$$\overrightarrow{a_k} = \frac{\overrightarrow{v_k} - \overrightarrow{v_{k-1}}}{T} \tag{6-30}$$

Both velocity vectors in (6-30) are estimated velocities. After the measurement and estimation of the velocity vector, the acceleration vector is estimated considering the acceleration derivative vector as an input in (6-27).

Table 9: Kalman filter 3

$$\begin{bmatrix} r_k^x \\ r_k^y \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{k-1}^x \\ r_{k-1}^y \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v_{k-1}^x \cdot T + a_{k-1}^x \cdot \frac{T^2}{2} + da_{k-1}^x \cdot \frac{T^3}{6} \\ v_{k-1}^y \cdot T + a_{k-1}^y \cdot \frac{T^2}{2} + da_{k-1}^y \cdot \frac{T^3}{6} \end{bmatrix} + w_{k-1}^r \tag{6-23}$$

$$\begin{bmatrix} z_k^{r,x} \\ z_k^{r,y} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_k^x \\ r_k^y \end{bmatrix} + v_k^r \tag{6-24}$$

$$\begin{bmatrix} v_k^x \\ v_k^y \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v_{k-1}^x \\ v_{k-1}^y \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} a_{k-1}^x \cdot T + da_{k-1}^x \cdot \frac{T^2}{2} \\ a_{k-1}^y \cdot T + da_{k-1}^y \cdot \frac{T^2}{2} \end{bmatrix} + w_{k-1}^v \tag{6-25}$$

$$\begin{bmatrix} z_k^{v,x} \\ z_k^{v,y} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v_k^x \\ v_k^y \end{bmatrix} + v_k^v \tag{6-26}$$

$$\begin{bmatrix} a_k^x \\ a_k^y \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} a_{k-1}^x \\ a_{k-1}^y \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} da_{k-1}^x \cdot T \\ da_{k-1}^y \cdot T \end{bmatrix} + w_{k-1}^a \tag{6-27}$$

$$\begin{bmatrix} z_k^{a,x} \\ z_k^{a,y} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} a_k^x \\ a_k^y \end{bmatrix} + v_k^a \tag{6-28}$$

$$da_{k-1}^i = \frac{a_{k-1}^i - a_{k-2}^i}{T} \tag{6-29}$$

6.3.1.5 Filter 4

The fourth filter was designed to estimate the position and velocity of the target using two independent Kalman filters, one regarding x and the other regarding y directions. In the process model, the accelerations in both axes are modeled as noisy disturbances. The model of the fourth filter is shown in table 10. The index *i* represents one of the directions, x or y.

Table 10: Kalman filter 4

$$\begin{bmatrix} r_k^i \\ v_k^i \end{bmatrix} = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{k-1}^i \\ v_{k-1}^i \end{bmatrix} + \begin{bmatrix} T^2/2 \\ T \end{bmatrix} \cdot w \tag{6-31}$$

$$\begin{bmatrix} z_k^{r,i} \\ z_k^{v,i} \end{bmatrix} = \begin{bmatrix} 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} r_k^i \\ v_k^i \end{bmatrix} + v \tag{6-32}$$

6.3.1.6 Filter 5

The fifth filter was designed to estimate the position, velocity and acceleration of the target using two independent Kalman filters, one regarding x and the other regarding y directions. Analogously as the fourth filter, in the process model, the acceleration derivative in both axes is modeled as noisy disturbances. The model of the fifth filter is shown in table 11. The index *i* represents one of the directions, x or y.

Table 11: Kalman filter 5

$$\begin{bmatrix} r_k^i \\ v_k^i \\ a_k^i \end{bmatrix} = \begin{bmatrix} 1 & T & T^2/2 \\ 0 & 1 & T \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{k-1}^i \\ v_{k-1}^i \\ a_{k-1}^i \end{bmatrix} + \begin{bmatrix} T^3/6 \\ T^2/2 \\ T \end{bmatrix} \cdot w \tag{6-33}$$

$$\begin{bmatrix} z_k^{r,i} \\ z_k^{v,i} \\ z_k^{a,i} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} r_k^i \\ v_k^i \\ a_k^i \end{bmatrix} + v \tag{6-34}$$

6.3.1.7 Evaluation of Filters

The filters designed were evaluated with respect to estimation accuracy and computational efficiency. Several tests were implemented in order to provide the means of evaluation. The purpose of such evaluation is selecting the best filters to be added to the Object Tracking behavior and tested with data from the VC.

For the estimation accuracy tests, five mathematical models for the target's motion were simulated. In each model, the position of the target followed a stochastic process as shown in (6-36).

$$R(t) = f(t) + W'_p(t) \tag{6-35}$$

$$\hat{R}(t) = R(t) + W_m(t) \tag{6-36}$$

In (6-35), *R(t)* represents the stochastic process describing the target's position, composed by the sum of a deterministic vector function *f(t)* and a process disturbance modeled as the stochastic process $W'_p(t)$. In (6-36), $\hat{R}(t)$ represents the stochastic process describing the measurement of the target's position, composed by the sum of the actual target position *R(t)* and a measurement noise modeled as the stochastic process $W_m(t)$.

The filter was set to estimate the target's position and accuracy tests were realized. Such tests were separated in two parts. In the first part, the accuracy of the estimation of the target's position was tested. In the second part, the focus was testing the filters' accuracy when predicting the future position of the target.

For each model of target motion, the filters were used in several tests to estimate and predict the position of the target. In each test, the simulation of target movement and tracking was run for several "tunings" of process noise covariance and measurement noise covariance. The result of each test consisted in the smallest absolute accumulated error provided by the tuning of the filter which minimizes such error. The tests differ between themselves on the range of the measurement and process noise.

Resuming, the evaluation was separated in two parts in which every filter was tested, each part of the evaluation was separated in several different target motion models, each model was used in several tests and each test was run with several tunings. Table 12 presents the settings of the evaluation.

Table 12: Settings of filters' evaluation

Entity	Setting
Number of Target Motion Models	5
Cycle period	20 ms
Number of tests for each model[†]	209
Cycles of prediction[‡]	10 cycles
Variety of tunings (Filters 1, 2 and 3)	37
Variety of tunings (Filters 4 and 5)	49
Cycles for each simulation	500

6.3.1.8 Target Movement Models

This subsection presents the models used for the simulation of a moving target. The measurement noise in the evaluation is represented by a stochastic process $\overrightarrow{W_m}(t) : \mathbf{R} \rightarrow [-W_{m,\max}, W_{m,\max}] \times [-W_{m,\max}, W_{m,\max}]$ such that the probability density function (*pdf*) in each instant *t* is uniformly distributed in the intervals $[-W_{m,\max}, W_{m,\max}]$. The value of $W_{m,max}$ varies from test to test, and it is such that $W_{m,\max} \in \{0.01, 0.02, ..., 0.09, 0.10\}$.

With the exception of the fifth model, the process noise in the evaluation is represented by the stochastic process $\overrightarrow{W_p}(t) : \mathbf{R} \rightarrow [-W_{p,\max}, W_{p,\max}] \times [-W_{p,\max}, W_{p,\max}]$ such that the pdf in each instant *t* is uniformly distributed in the intervals $[-W_{p,\max}, W_{p,\max}]$. The value of $W_{p,max}$ varies from test to test, and the range of values which $W_{p,max}$ can assume varies according to the target motion model.

The variation on the range of values defined by $W_{p,max}$ and $W_{m,max}$ which the noise abides has the purpose of verifying how do each filter performs when the influence of the process and measurement noises increase or decrease.

The first model consists in a system influenced by a disturbance represented as a random acceleration derivative. Therefore, for given values at cycle *k-1* for *R[k-1]* (position), *V[k-1]* (velocity) and *A[k-1]* (acceleration), the target's position in cycle *k* is given by the

[†] Each test has a distinct configuration of measurement and process noise range, e.g. Test 11 has $W_{p,max} = 0.05$ and $W_{m,max} = 0.09$, test 12 has $W_{p,max} = 0.10$ and $W_{m,max} = 0.09$.

[‡] Cycles of prediction refers to the number of cycles ahead in the prediction accuracy part of the evaluation. With the number of cycles of prediction set to 10, the filter predicts the position of the target 10 cycles ahead and the absolute error of such prediction is recorded.

process described by (6-37). The maximum absolute value $W_{p,max}$ which the process noise can assume varies in each test of this model with the values $W_{p,\max} \in \{0.05, 0.25, 0.50, 0.75, ..., 4.50, 4.75, 5.00\}$. Equation (6-37) describes the discreet motion process, with T representing the cycle period.

$$\vec{R}[k] = \vec{R}[k-1] + \vec{V}[k-1] \cdot T + \vec{A}[k-1] \cdot T^2/_2 + [(1,1) + \overrightarrow{W_p}[k-1]] \cdot T^3/_6 \quad (6\text{-}37)$$

The second model consists in a system influenced by a disturbance in the acceleration of the target. The acceleration is a sum of a deterministic function *a[i]* and the stochastic process $W_p[i]$. Therefore, for given conditions *R[k-1]* (position) and *V[k-1]* (velocity), the target's position in a given cycle k is given by the process described by (6-38). The maximum absolute value $W_{p,max}$ which the process noise can assume varies in each test of this model with the values $W_{p,\max} \in \{0.01, 0.05, 0.10, 0.15, ..., 0.90, 0.95, 1.0\}$. Equations (6-38) and (6-39) describe the discreet motion process, with i representing the cycle.

$$\vec{R}[k] = \vec{R}[k-1] + \vec{V}[k-1] \cdot T + \vec{A}[k-1] \cdot T^2/_2 \quad (6\text{-}38)$$

$$\vec{A}[i] = (1,1) \cdot [0.1213 \cdot T^3 \cdot i^3 - 1.8390 \cdot T^2 \cdot i^2 + 7.7585 \cdot T \cdot i] + \overrightarrow{W_p}[i] \quad (6\text{-}39)$$

Figure 62 illustrates the deterministic part *a[k]* of the acceleration process *A[k]* and a realization of the process with presence of noisy disturbance.

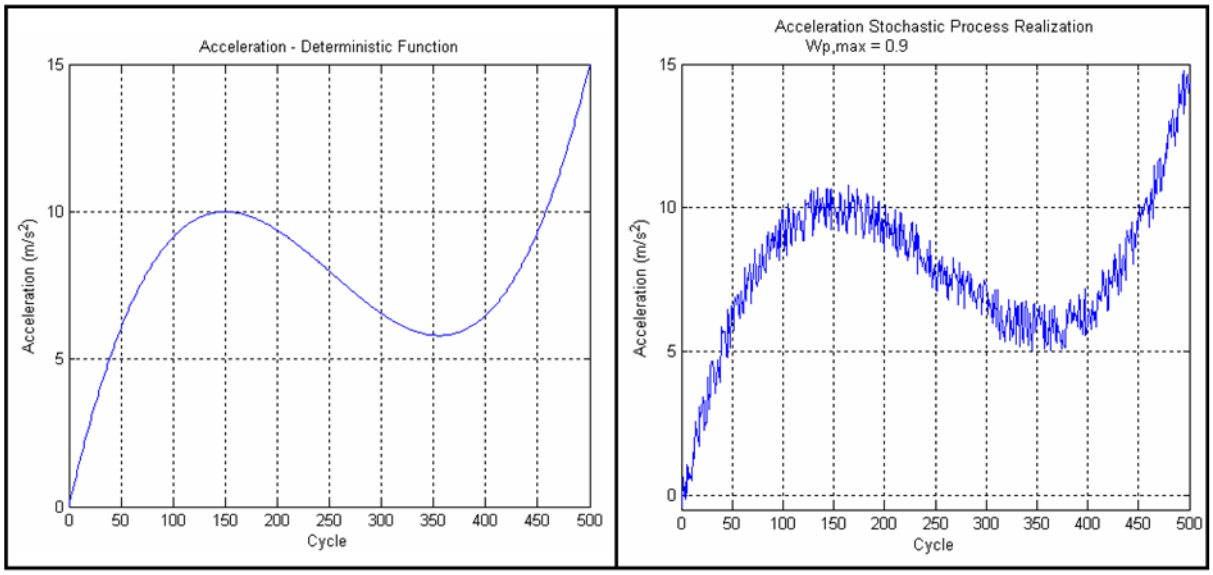

Figure 62: Acceleration process for motion model 2

The third model consists in a system influenced by a disturbance in the acceleration of the target. The acceleration has initial value (2,2) m/s^2 and is increased each cycle by the value of the stochastic process $W_p[k]$. Therefore, for given conditions *R[k-1]* (position) and *V[k-1]* (velocity), the target's position in a certain cycle *k* is given by the process described by (6-40). The maximum absolute value $W_{p,max}$ which the process noise can assume varies in each test of this model with the values $W_{p,\max} \in \{0.01, 0.05, 0.10, 0.15, ..., 0.90, 0.95, 1.0\}$. The discreet motion process is described by equations (6-40) and (6-41).

$$\vec{R}[k] = \vec{R}[k-1] + \vec{V}[k-1] \cdot T + \vec{A}[k-1] \cdot T^2/2 \tag{6-40}$$

$$\vec{A}[i] = \vec{A}[i-1] + \vec{W_p}[i], \ \vec{A}[0] = (2,2)\, m/s^2 \tag{6-41}$$

The fourth model consists in a system influenced by a disturbance in the acceleration of the target. The acceleration has initial value (2,2) m/s^2 and in each cycle assumes the value of the stochastic process $W_p[k]$. Therefore, for given conditions *R[k-1]* (position) and *V[k-1]* (velocity), the target's position in a cycle *k* is given by the process described by (6-42). The maximum absolute value $W_{p,max}$ which the process noise can assume varies in each test of this model with the values $W_{p,\max} \in \{0.01, 0.05, 0.10, 0.15, ..., 0.90, 0.95, 1.0\}$. The discreet motion process is described in (6-42).

$$\vec{R}[k] = \vec{R}[k-1] + \vec{V}[k-1] \cdot T + \vec{W}_p[k-1] \cdot T^2/2 \tag{6-42}$$

The fifth model consists in a system influenced by a disturbance in the acceleration of the target. The acceleration obeys an "on-off" rule, with probabilities $P_{on}[i] = P_{off}[i] = 0.5$. In each cycle when the acceleration is on, it assumes one of the values of $\{ -W_{p,max}, W_{p,max}\}$ with probability 0.5. Therefore, for given conditions *R[k-1]* (position) and *V[k-1]* (velocity), the target's position in a cycle *k* is given by the process described by (6-43). The maximum absolute value $W_{p,max}$ which the process noise can assume varies in each test of this model with the values $W_{p,\max} \in \{0.1, 0.5, 1.0, 1.5, ..., 9.0, 9.5, 10.0\}$. The discreet motion process is described by equations (6-43) and (6-44).

$$\vec{R}[k] = \vec{R}[k-1] + \vec{V}[k-1] \cdot T + \vec{W_p}[k-1] \cdot T^2/2 \tag{6-43}$$

$$\begin{cases} P_{W_p[i]=W_{\max}} = P_{W_p[i]=-W_{\max}} = 0.25 \\ P_{W_p[i]=0} = 0.5 \end{cases} \quad (6\text{-}44)$$

6.3.1.9 Filter Tests Results

As explained before, the filters were submitted for testing when tracking a target following the motion models shown in the previous subsection. The tests concerned the accuracy of each filter's target position estimation, target future position prediction and also the computational efficiency of the filters.

In the target position estimation accuracy tests, in each test a filter would be used to estimate the current position of the target following one of the five models. The tests were realized such that for each filter, the same target motion model would be tested using several tunings. The accumulated absolute error of estimation was used to evaluate the filters' performances. For each filter and target motion model, the results of the tuning which provided the lowest estimation error would be recorded.

The results of the position estimation accuracy tests using each are provided in figures 63, 64, 65, 66 and 67. In the graphs, the term *Test ID* refers to a specific configuration of process and measurement noise range. With respect to each filter, the accumulated error of each test relates to the tuning which provided the smallest error.

As presented on the figures 63, 64, 65, 66 and 67, filters 4 and 5 consistently outperformed filters 1, 2 and 3. In the fifth model of target motion, with "on-off" acceleration model, filter 4 presented better results than filter 5. In the tests of the fourth model, filters 4 and 5 had similar performances. In the other models, filter 5 provided most accurate estimations.

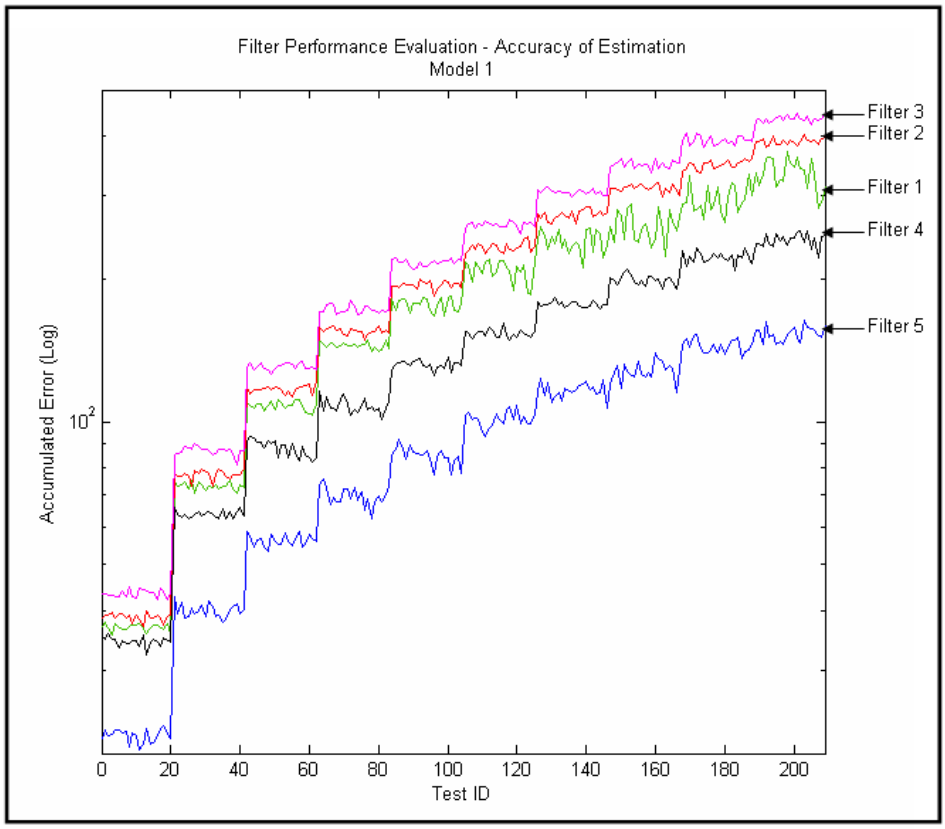

Figure 63: Accuracy of estimation evaluation for target motion model 1

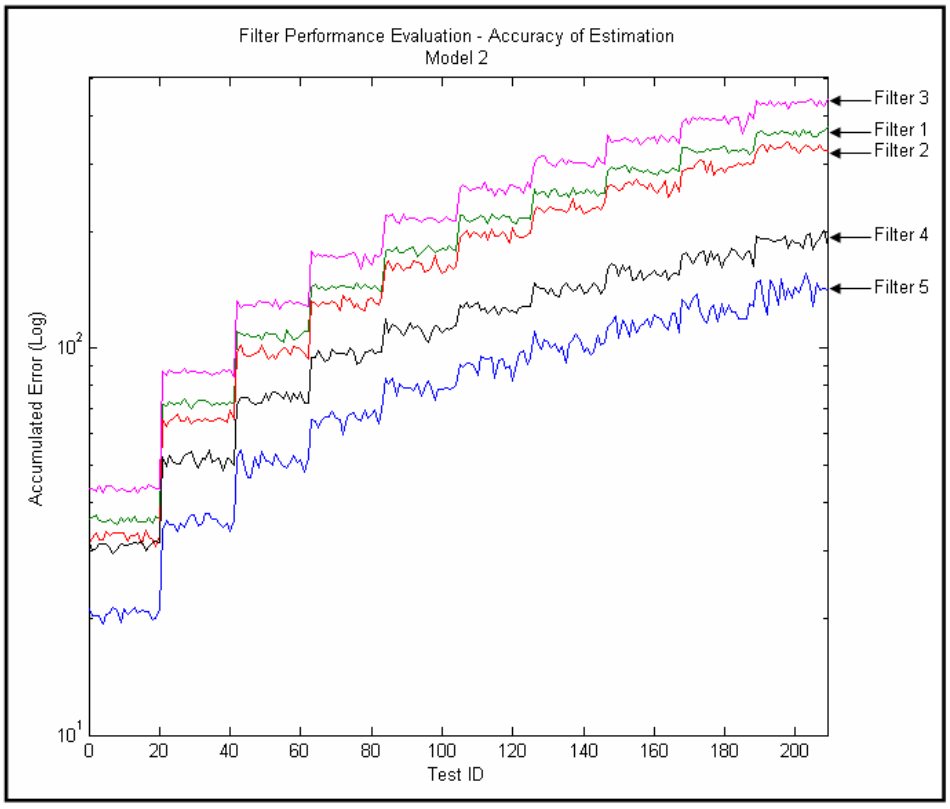

Figure 64: Accuracy of estimation evaluation for target motion model 2

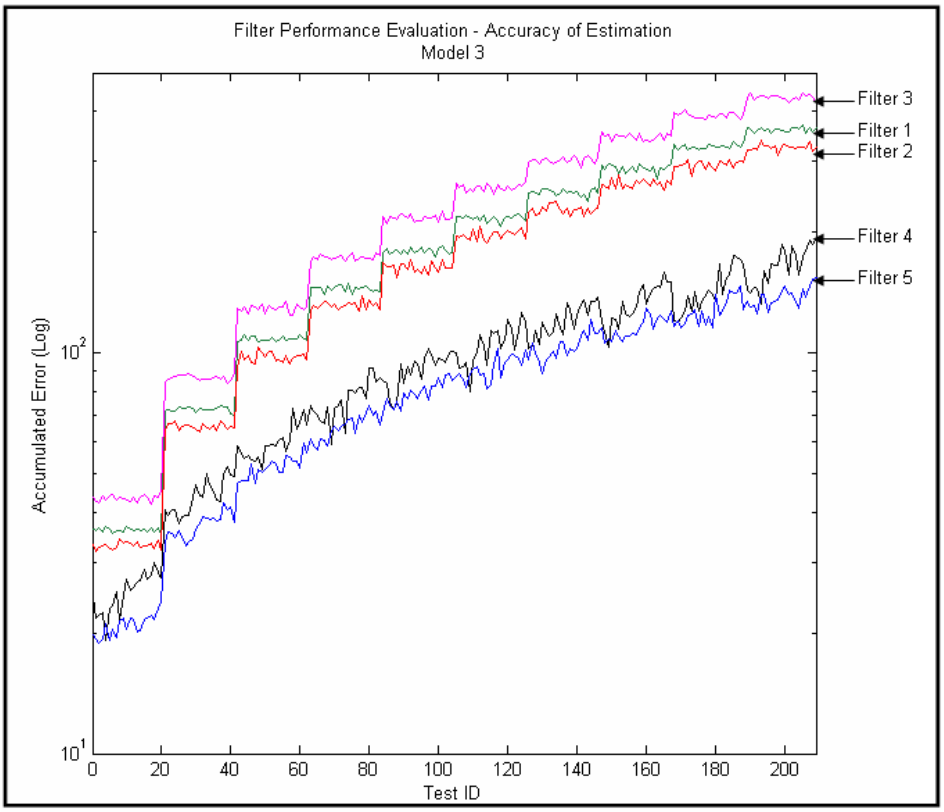

Figure 65: Accuracy of estimation evaluation for target motion model 3

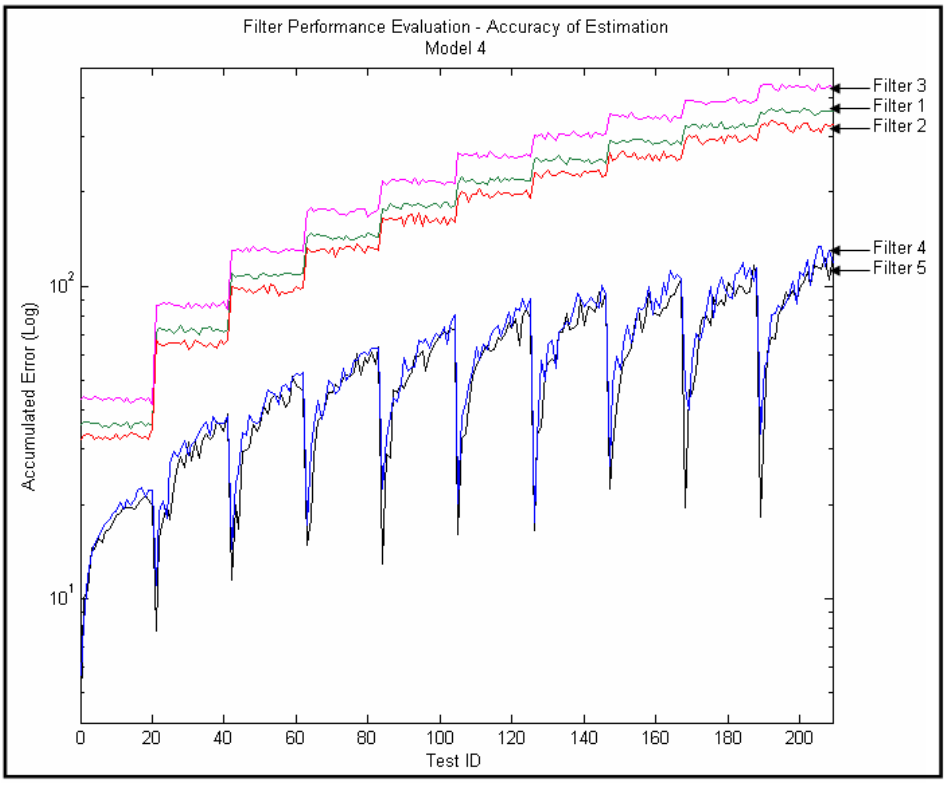

Figure 66: Accuracy of estimation evaluation for target motion model 4

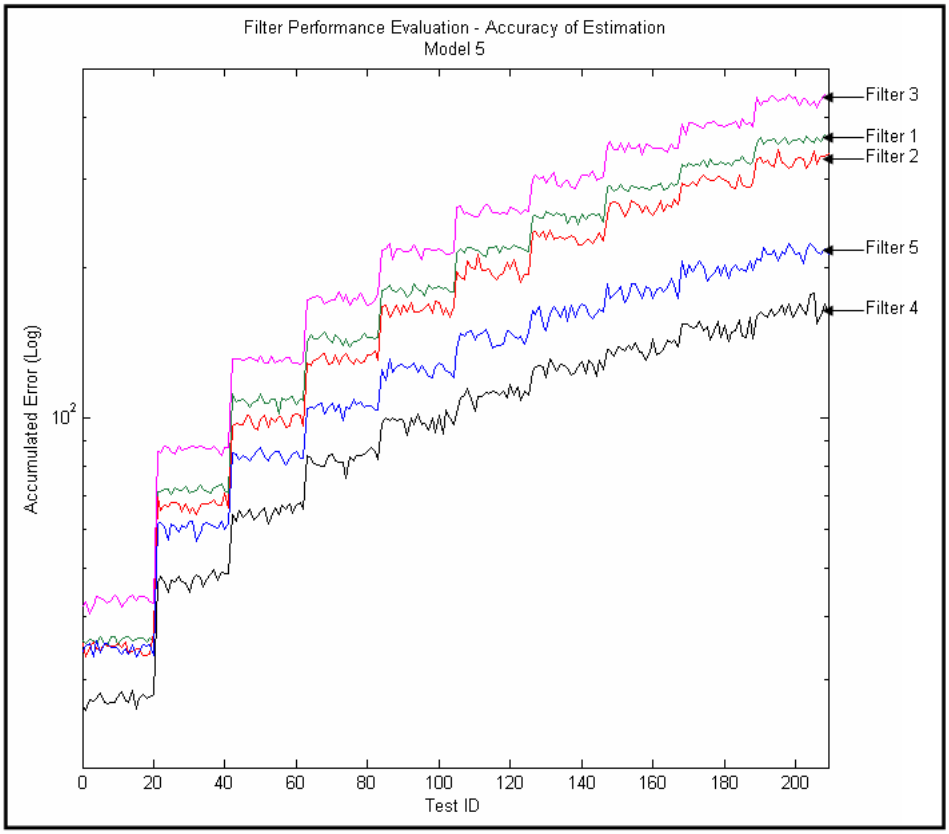

Figure 67: Accuracy of estimation evaluation for target motion model 5

Forth, the tests conceived to evaluate the accuracy of the filters when predicting the future position of the target are presented. The tests were configured such that the prediction would be made for 10 cycles ahead. Every ten cycles, the absolute error of the prediction was recorded and added to the accumulated absolute error for the test. Besides this peculiarity, the tests in this case functioned in the same fashion as explained before in this subsection. Figures 68, 69, 70, 71 and 72 present the results of the position prediction accuracy tests.

As presented on figures 68, 69, 70, 71 and 72, filters 4 and 5 once again outperformed filters 1, 2 and 3 in most cases. As in the first set of tests, in the fifth model of target motion filter 4 presented better results than filter 5, and in the fourth model, the two filters had similar performances. In the other models, filter 5 provided most accurate estimations. In the first model, filter 1 and filter 4 presented results of resembling accuracy. In the second model, filter 2 outperformed filter 1 in the first 26 tests, which corresponded to tests with low measurement noise. As the measurement noise grew in the following tests, filter 4 gradually outperformed filter 2.

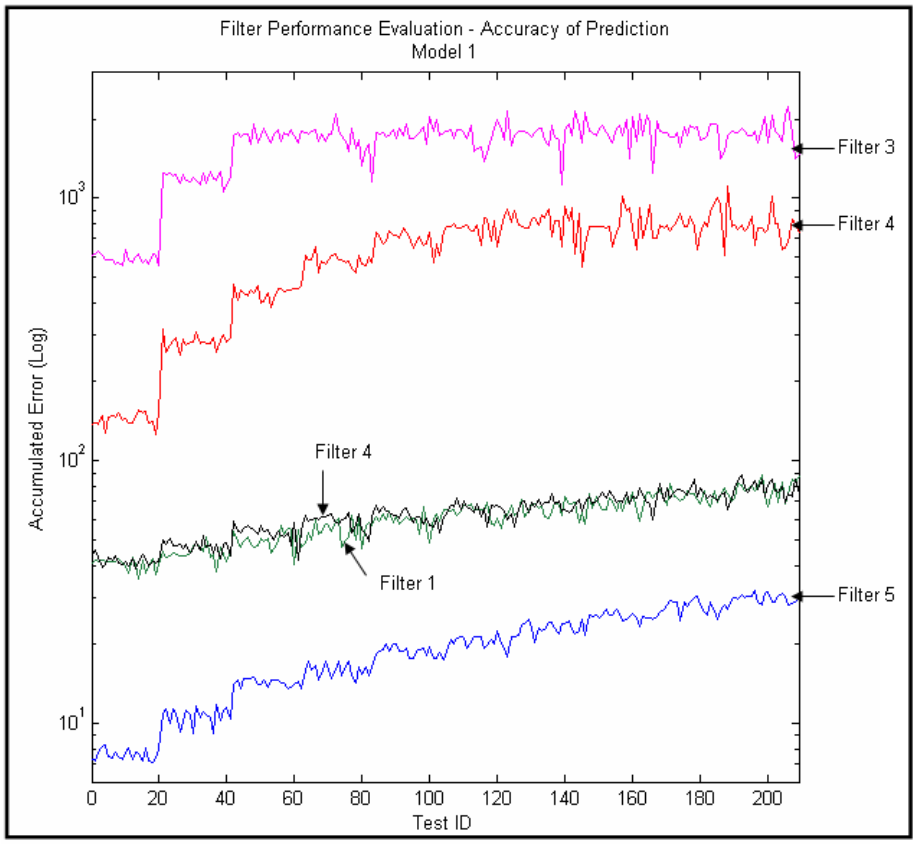

Figure 68: Accuracy of prediction evaluation for target motion model 1

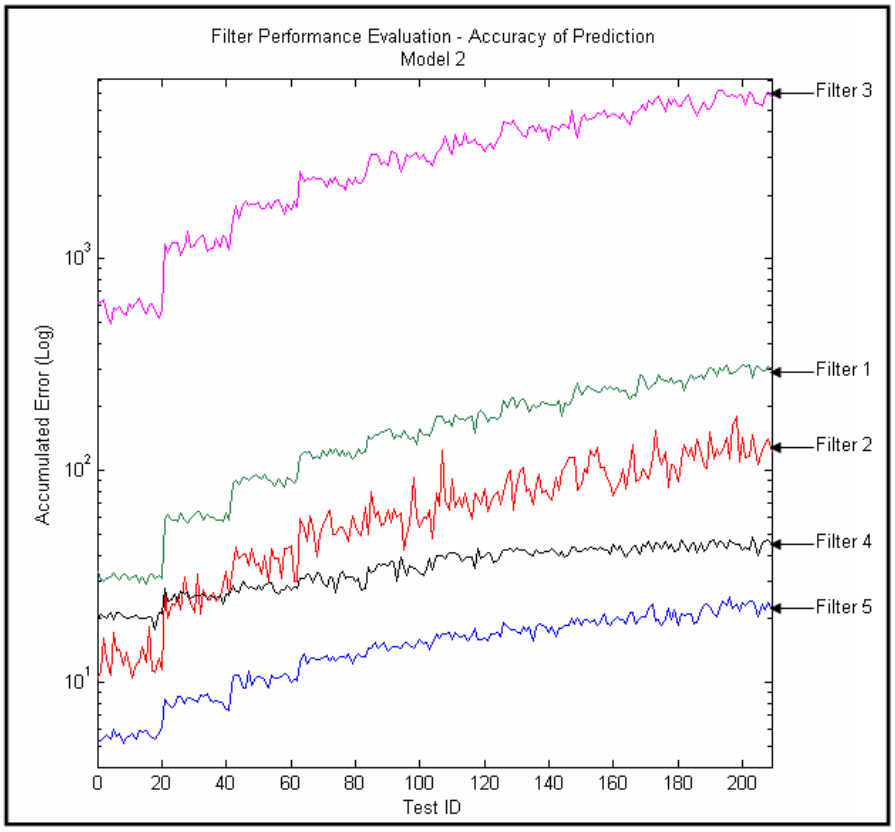

Figure 69: Accuracy of estimation evaluation for target motion model 2

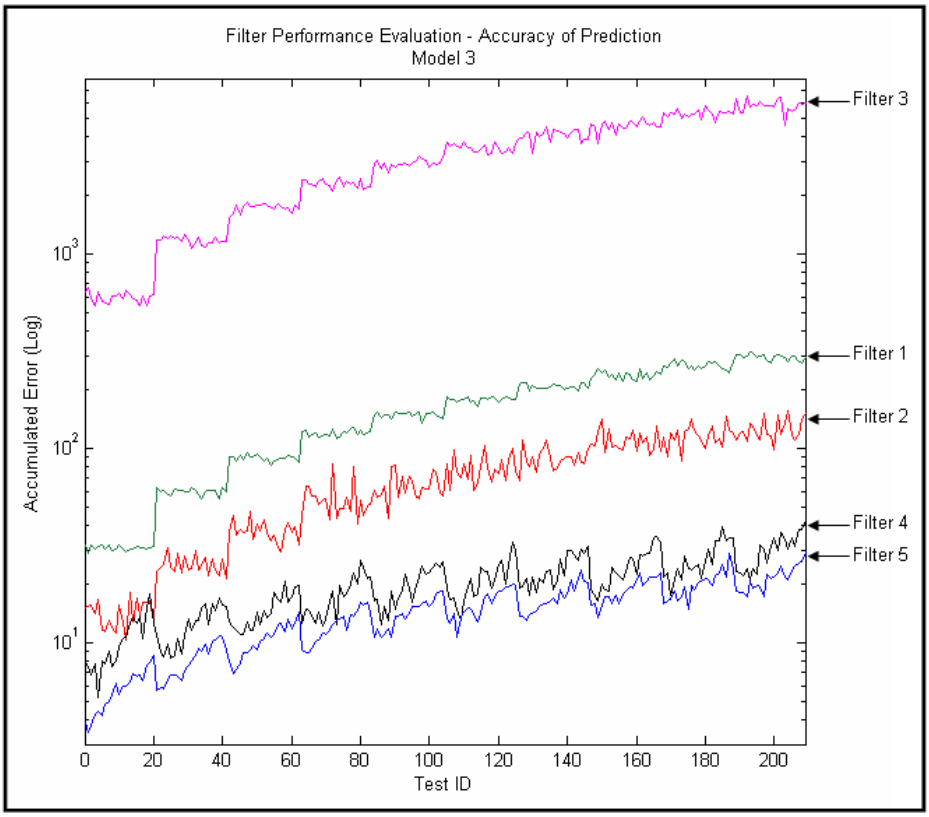

Figure 70: Accuracy of estimation evaluation for target motion model 3

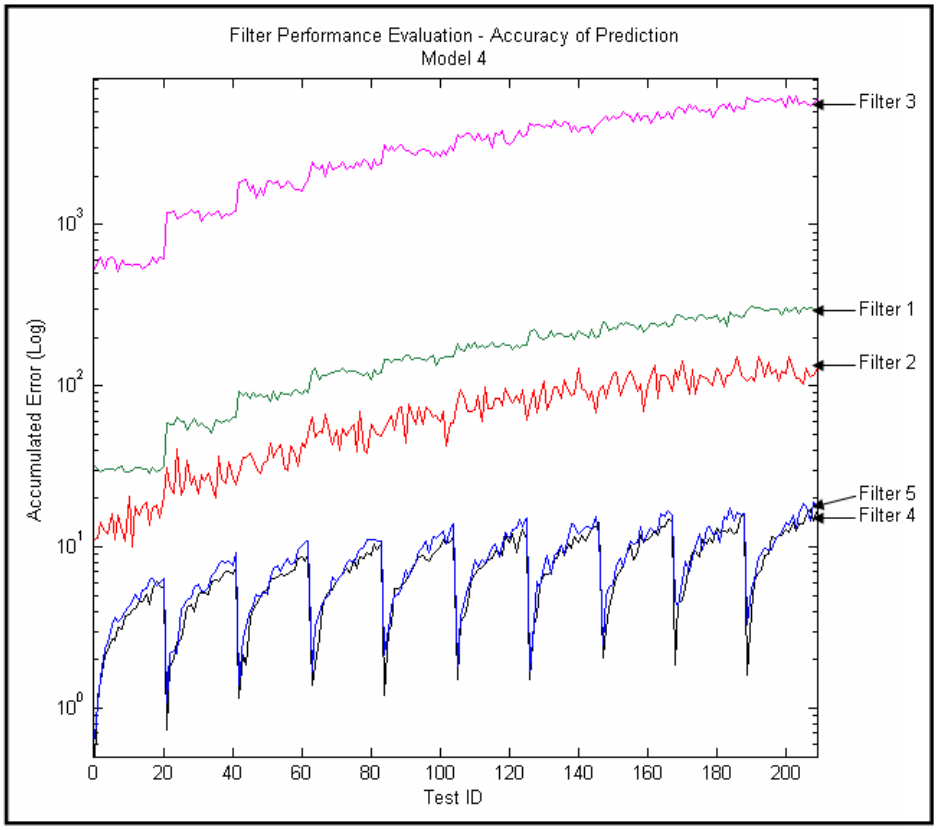

Figure 71: Accuracy of estimation evaluation for target motion model 5

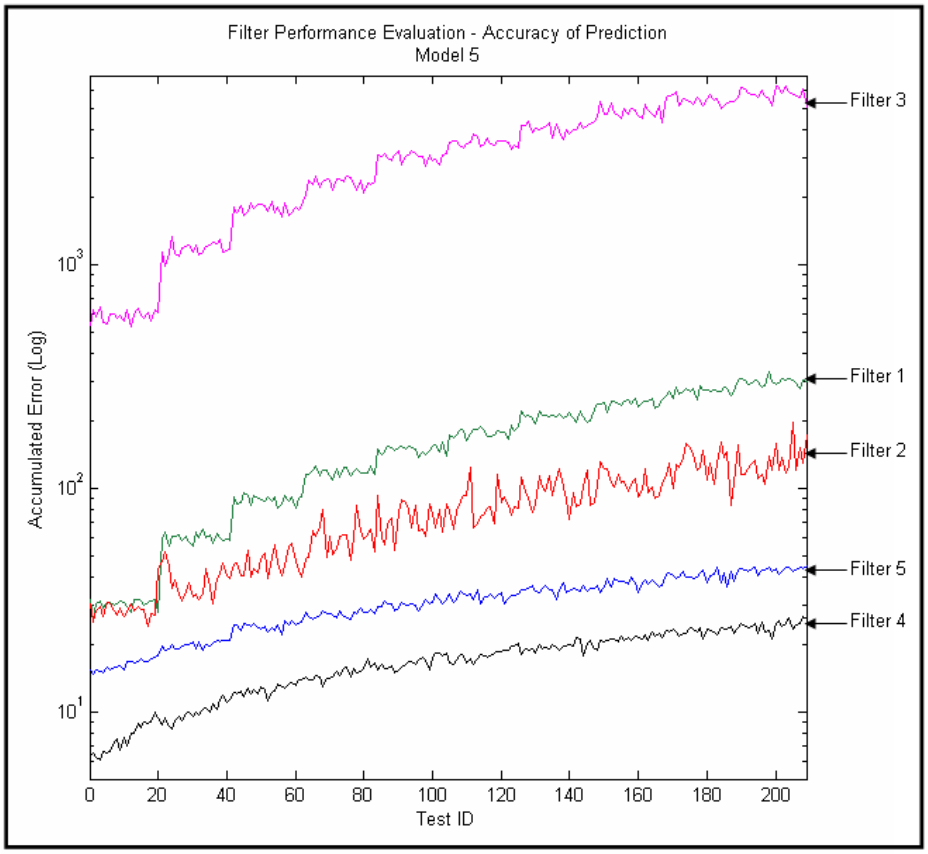

Figure 72: Accuracy of estimation evaluation for target motion model 5

The third set of tests concerned the computational efficiency of the filters. In this test, 2 million cycles of a process were simulated. In each cycle the filters were used to estimate the process' state and predict the state three cycles ahead. The total computational time was measured for each filter. Figure 73 presents the results of such test.

As shown in figure 73, filter 1 has shown superior computation efficiency, but its performance was similar to those of filters 4 and 5. Filters 2 and 3 were considerably less efficient.

The overall accuracy performances of filter 4 and 5 in the tests were considerably better when compared to the others. Furthermore, their computational efficiency was almost as good as the best result obtained (filter 1), with both filters requiring less than 15% more computational time. Therefore, filters 4 and 5 were added to the Object Tracking behavior. Their performance was also tested using HITL simulation. Such tests are presented on section **6.5**. The filters 4 and 5 will forth be referred to as, respectively, *Filter PV* (estimates position and velocity) and *Filter PVA* (estimates position, velocity and acceleration).

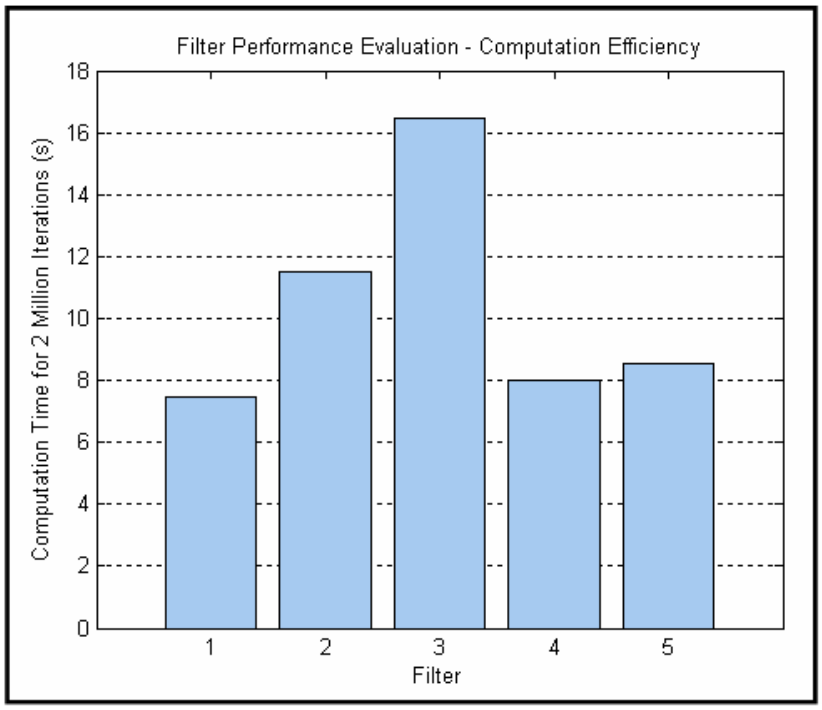

Figure 73: Filters' computational efficiency evaluation

6.3.2 Target Pursuit Strategy

As presented on section **6.2**, the target pursuit strategy developed in this work consisted, broadly put, in commanding the UAV to move to predicted future positions of the target using an adapted waypoint navigator. This section presents the development of the pursuit strategy.

6.3.2.1 Adapted waypoint Navigator

The waypoint navigator is adapted such that the system uses a Hover To behavior whose destination is, in each cycle, modified according to the filter's prediction on the target's future position. Therefore, in each cycle the Hover To behavior produces the output signal to reach the waypoint selected by the Object Tracking behavior. Figure 74 illustrates the principle.

As seen on figure 74, the Object Tracking behavior sets a different waypoint "target" every cycle, considering the expected future position of the object in a number of cycles ahead.

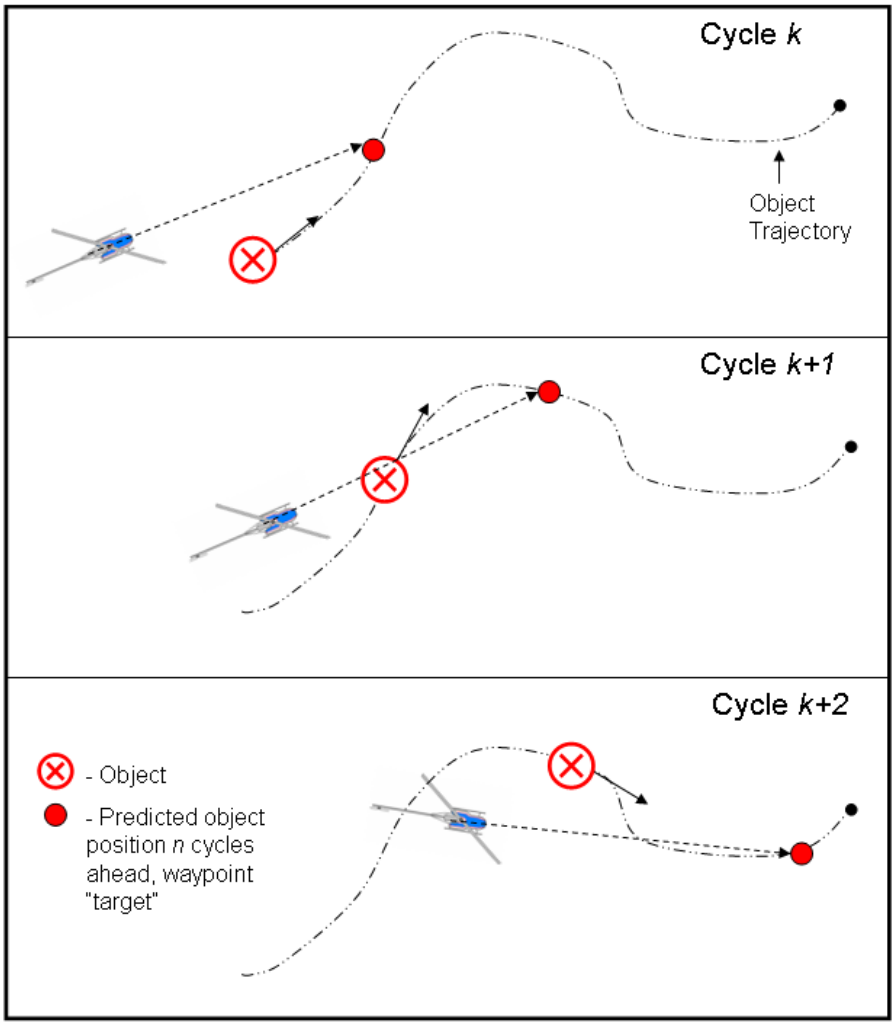

Figure 74: Adapted waypoint navigator

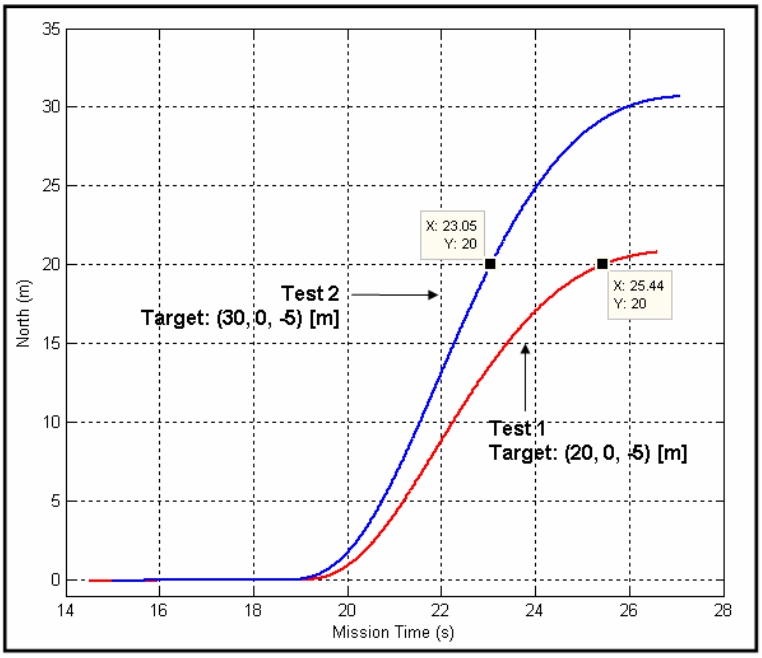

Figure 75: Response time

As stated before, the system responds more vigorously and quickly when commanded to hover to a position further away. Therefore, commanding the UAV to hover to a predicted future position of the object would increase the distance between waypoint target and UAV, thus improving the response time in the critical situation when the target is moving *away* from the helicopter. Figure 75 presents results of a SITL simulation confirming such statement.

In the SITL simulation, the helicopter started at position (0, 0, -5) [m]. The intention was comparing the time to reach coordinates (20, 0, -5) [m] when the UAV is commanded to hover directly to such point, and when it is commanded to hover further to point (30, 0, -5) [m]. As shown on figure 75, the time taken for the helicopter to reach the coordinates is decreased in almost 40% when the waypoint target is increased in 10 [m].

Since in the Object Tracking's waypoint navigator the objective is moving in the direction of the waypoint target (and not necessarily reaching it), the strategy favors a faster velocity when pursuing the target.

6.3.2.2 Prediction Time

As the pursuit strategy consists in choosing waypoint targets corresponding to predicted future positions of the ground object, the next step in the design of the motion strategy is determining how far ahead in the future are such predictions made.

In order to choose an appropriate prediction time, the following criteria have been established.

1. The aircraft should be able to quickly approach the ground object;
2. After reaching a close position relative to the ground object, the aircraft should not overcome the object or loose its sight because of the UAV's motion.

In the context of the pursuit strategy proposed in this work, the criteria 1 and 2 are concurrent. With a high prediction time, the aircraft responds more vigorously when its relative distance to the object is large, therefore reducing the approach time, but it enhances the chance of overcoming the ground object. On the other hand, a low prediction time is better suited to accompany the object's motion, but it increases the approach time and the chance of not providing the necessary agility and speed to reach the object. Figure 76 presents data acquired via HITL simulation which confirm such statements.

In the SITL simulation, ARTIS was commanded to track a simulated visual input of a target moving in a straight line. In the first simulation, the prediction time used was high (50 [ms]), and in the second we used a short prediction time (10 [ms]). As seen, in the first case

the helicopter quickly overcomes the target. In the second case, the aircraft is not capable of reaching the target. Neither situation is desired.

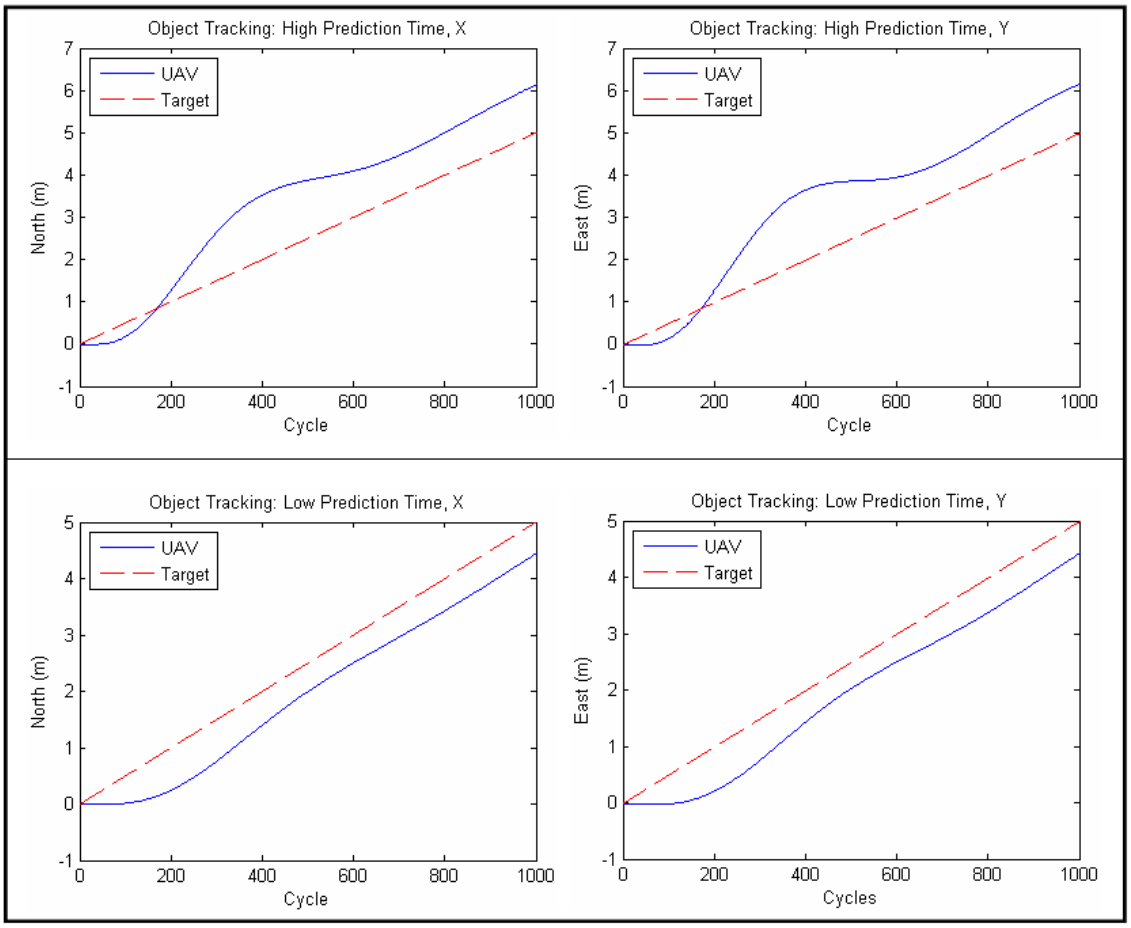

Figure 76: Prediction time effect over pursuit

To address this problem, we proposed a pursuit strategy divided into two tracking modes, named *approach mode* and *accompany mode*. When in approach mode, the Object Tracking behavior uses a high prediction time T_{high}, increasing the aircraft's speed and reactivity. When in accompany mode, the Object Tracking behavior uses a low prediction time T_{low}, providing a balanced response to the objects motion. Figure 77 presents the situations in which each mode is activated.

As the Object Tracking behavior applies position filtering in both X and Y directions, the modes are chosen independently for each filter (X or Y). As seen in figure 77, the accompany mode is activated in situations which the aircraft has overcome or is in danger of overcoming the ground object, providing thus a balanced response to the object's motion. The approach mode is activated in critical situations, such as when the object and the UAV are moving in opposite directions or when the distance between aircraft and ground object is too

large. In such situations, the approach mode provides a vigorous response to the target's motion, enhancing ARTIS' capability of reaching the object.

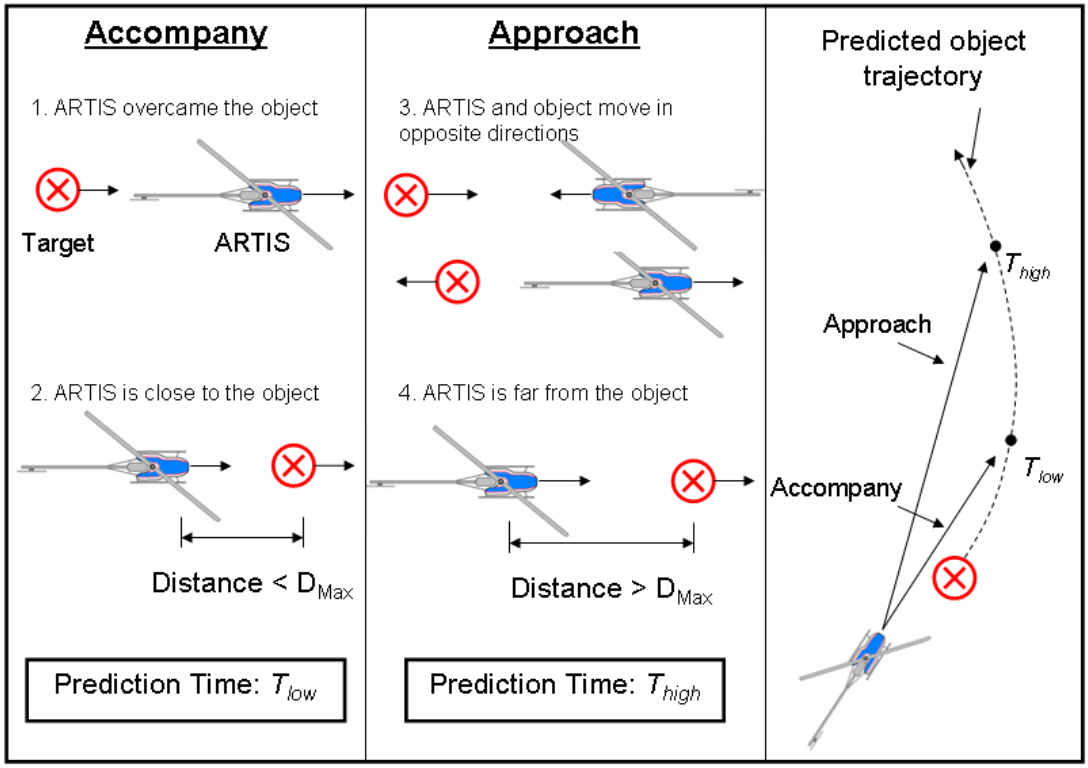

Figure 77: Accompany and Approach tracking modes

6.3.2.3 Pursuit Strategy Overview

In the previous subsections, we presented the several details which have been addressed in order to design the Object Tracking behavior's pursuit strategy. In the diagram of figure 78, we present the overview of such strategy, demonstrating how its aspects function together.

6.4 Management Sub-problem

As explained in section **6.2**, developing methods which address the problems presented by the different desirable functionalities of the Object Tracking behavior is not sufficient to guarantee the overall functionality of the behavior itself. It is necessary to provide the UAV's system the necessary means to apply the behavior, responding properly to external and internal events.

In this section, we present the decision making capabilities of the Object Tracking behavior, as well as relevant related topics. The management module takes account of safety and performance efficiency of the behavior.

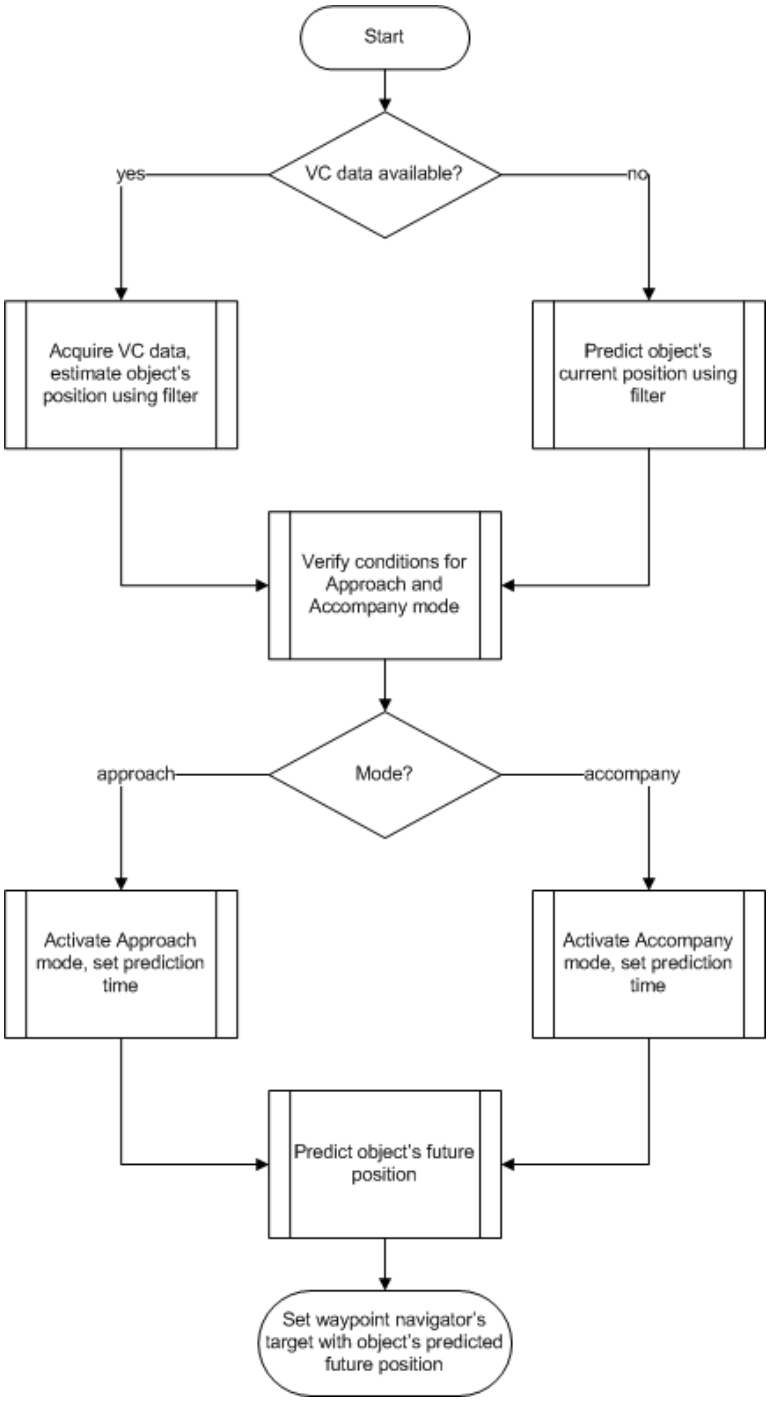

Figure 78: Pursuit strategy flowchart

Forth, the following terms will be used to denote different aspects of the Object Tracking behavior:

- *Object Tracking State*: the module responsible for managing the events which could take place when the Object Tracking behavior is activated and the different stages on the temporal evolution of the behavior. Therefore, this is the module designed to provide the behavior's decision making capabilities.
- *Object Tracking Behavior Object*: the module responsible for realizing the functionalities required by the Object Tracking behavior. This denomination is inspired by the implementation of the behavior using Object Oriented Programming.

6.4.1 Object Tracking Behavior Modeling

The Object Tracking behavior was modeled according to UML specifications for Statechart diagrams. This choice was based on the advantages of statechart-based modeling as presented on subsection **3.1.1**. This model was conceived with the purpose of managing possible events during a mission managed by the Object Tracking behavior, as well as allowing the UAV to execute different stages of the behavior.

As stated in section **3.1**, the Object Tracking state is part of the statechart diagram designed to model the Supervisor module. The system transits to Object Tracking behavior when the system is performing a search object task and the target is spotted for the first time.

Note that in this section we distinguish the meanings of the expressions *finding object* and *spotting object*. Spotting the object denotes every occasion in which the VC provides data with the object's position. Finding the object denotes the situation in which the VC has recognized the object more than once during a short period of time. In this situation, we assume that the object is in fact within camera view and discard the hypothesis of false detection.

Figure 79 presents the statechart model designed for the Object Tracking behavior.

The Object Tracking state comprises three sub-states, namely *Find Object*, *Perform racking* and *Ascend*. The default sub-state of Object Tracking state is Find Object sub-state. Bellow follows a brief description of each sub-state.

- *Find Object*: the system transits to this sub-state at the beginning of the activation of Object Tracking behavior, when the object is spotted for the first time. The aircraft is commanded to quickly fly in the direction of the position of the first object spot. Until the object is spotted a second time, the behavior does not consider the object to be found.

- *Perform Tracking*: the system transits into this sub-state when the object is spotted for the second time within a short time period, therefore being considered to be found. In this sub-state, the data from the VC is filtered by the Kalman filter, and the Object Tracking behavior applies the pursuit strategy to command the UAV's motion.

- *Ascend*: the system transits into this state when the object is lost or when either the object or the UAV evades the perimeter. The UAV is commanded to return to the search area (when necessary), hover to a higher height in order to increase camera view and wait to verify if the object returns to camera view within the search area perimeter.

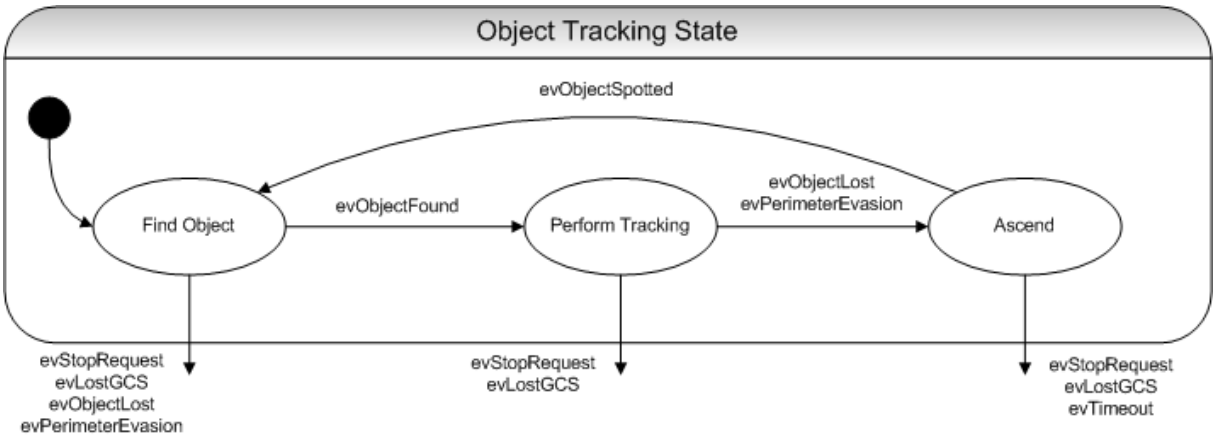

Figure 79: Object Tracking statechart model

The three sub-states presented are described in more detail in subsections **6.4.1.2** and **6.4.1.3**.

Below follows the description of the events relevant during while the system is in Fly Home state.

- *evStopRequest*: occurs when GCS issues a Stop command to the UAV;
- *evLostGCS*: occurs when the UAV's embedded system looses contact with GCS and the operator;
- *evObjectSpotted*: the VC provided data on the target's position, accusing the object to be within camera range;
- *evObjectFound*: the VC provided data on the target's position at least twice during a short time period (pre-defined and configurable), indicating that the object is within camera range and discarding the hypothesis of false detection;
- *evObjectLost*: occurs when ARTIS remains for a extended period of time (pre-defined and configurable) without spotting the target;

- *evPerimeterEvasion*: occurs when either the object or the aircraft leave the perimeter of the search area provided by the Search Object behavior;
- *evTmeout*: occurs during sub-state Ascend, when the object is not spotted within the search area for an extended time period (pre-defined and configurable).

As stated, one of the events which could occur while the Object Tracking state is active is the evasion of the search area perimeter by either the object or the aircraft. Therefore, before detailing the sub-states of Object Tracking state, we present a method used to check if a test point is within the search area.

6.4.1.1 Search Area Evasion Check

As stated in the previous chapter, the total search area is divided into convex cells. The first step into checking if the test point (i.e. UAV or ground object position) is within the search area is verifying if it is within any convex cell. Such problem can be related to the *point-in-polygon* computational geometry problem, which investigates whether a given point in the plane is located inside, outside, or on the boundary of a polygon[†].

The *Jordan Curve Theorem* is often used to assist in the solution of the point-in-polygon problem. The Theorem states that every non-self-intersecting loop in the plane divides the plane into an "inside" region and an "outside" region[‡]. With the Theorem, it is possible to prove is inside a polygon if, for any ray from this point, there is an odd number of crossings of the ray with the polygon's edges[†]. Figure 80 exemplifies such statement.

The approach to solve the problem addressed in this subsection involves using the Jordan Curve Theorem to check if the test point is within any of the convex cells. Therefore, we consider the ray on *x* positive direction from the test point and calculate how many times the ray crosses an edge of the convex cell.

Before applying such principle, a computation detail must be addressed. When one or more vertices are co-linear to the test point in the ray direction (*x* positive), the method shown above no longer applies. A simple solution consists into considering the test ray to be a half-plane divider, with one of the half-planes including each of the ray's points [51], [52]. The vertexes intersected by the test ray are always classified as being infinitesimally above the

[†] http://en.wikipedia.org/wiki/Point_in_polygon
[‡] http://en.wikipedia.org/wiki/Jordan_curve_theorem

ray. This solution grants that no vertices are intersected and the code is both simpler and speedier[†]. Figure 81 exemplifies this method.

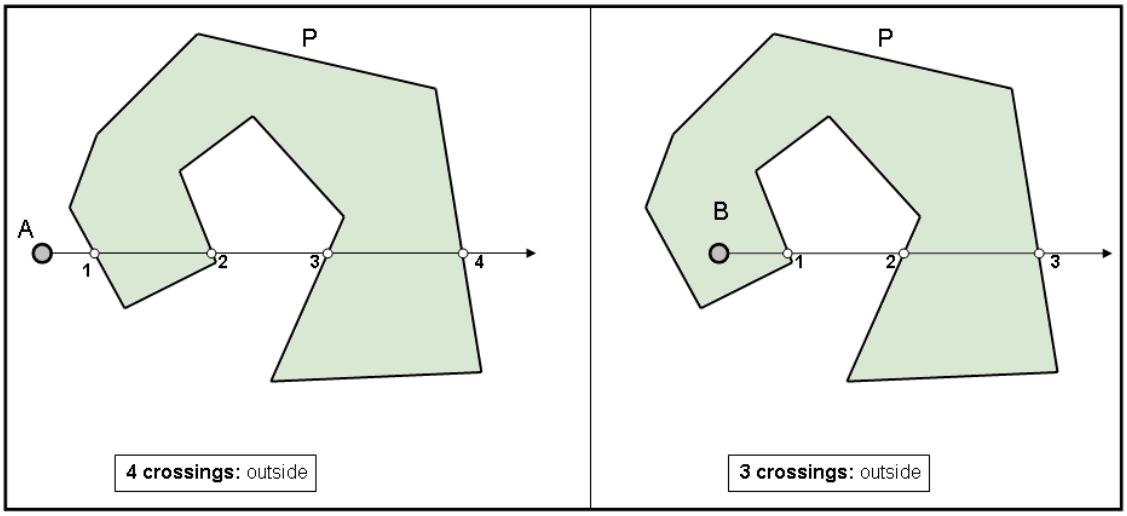

Figure 80: Jordan Curve Theorem

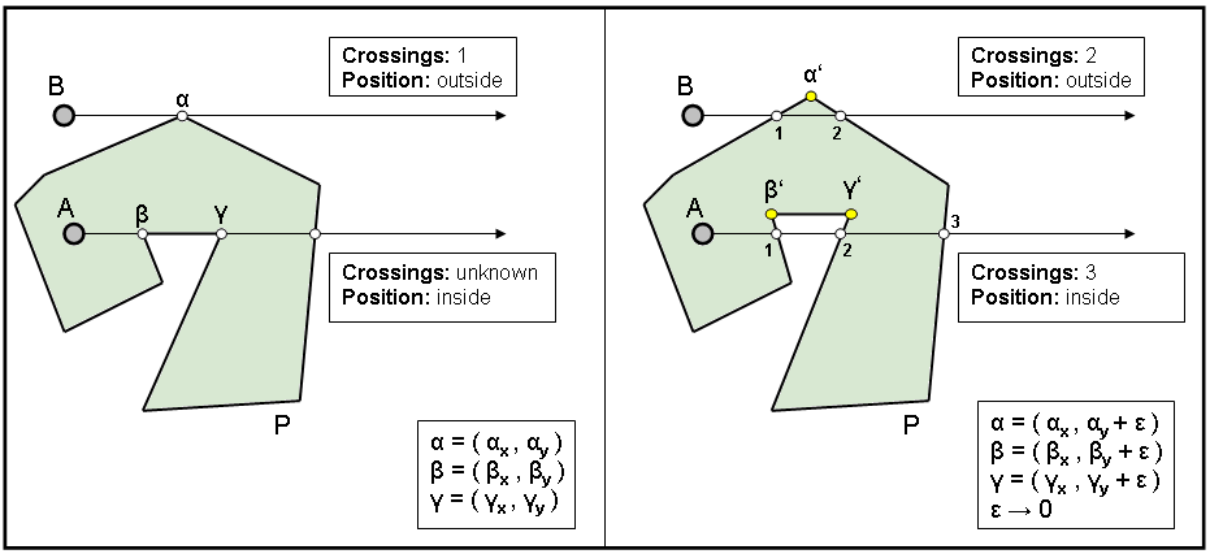

Figure 81: Adaptation of the crossing test

As presented before, the search area is divided into convex cells. Hence, there is the possibility that certain parts of the search area, namely "gaps" between cells, are not computed by the test shown before. To provide a complete solution it is necessary to verify if the aircraft is situated in a gap between cells which is contained by the search area.

[†] http://tog.acm.org/editors/erich/ptinpoly/

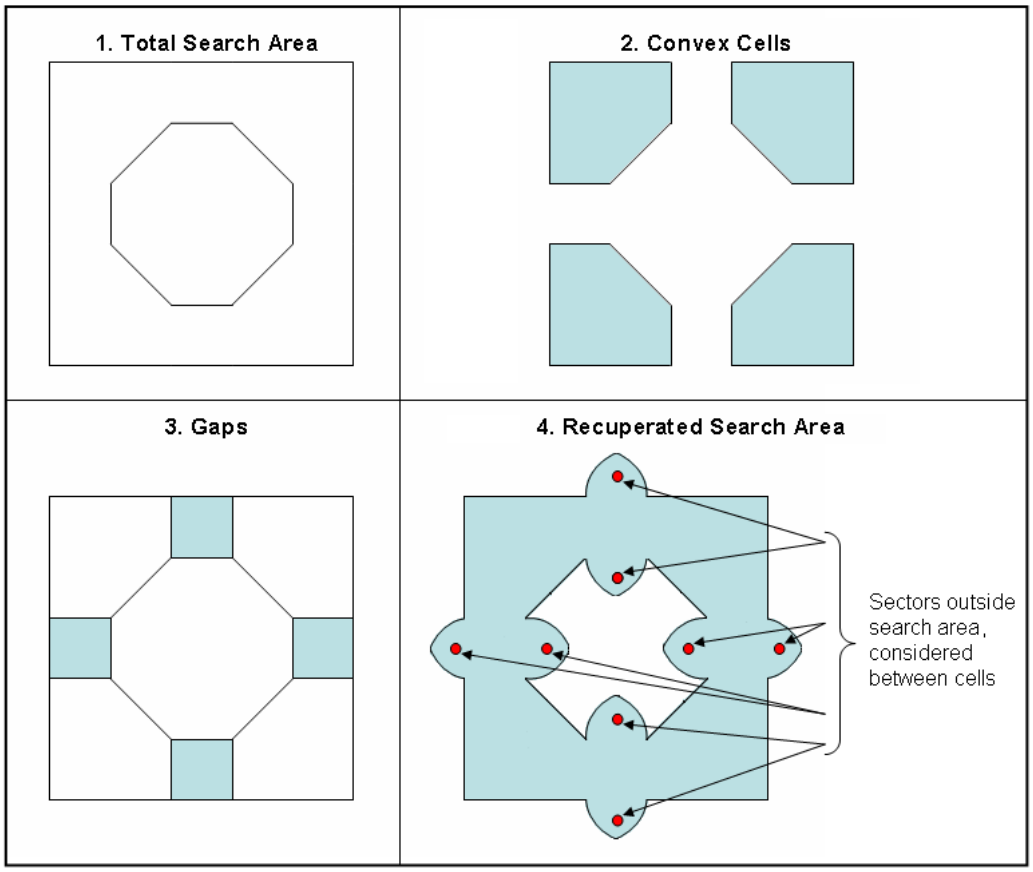

Figure 82: Search area as considered by the Object Tracking behavior

One simple and efficient solution is testing if the test point is located within a maximal distance (pre-defined and configurable) from two or more cells. The disadvantage of this method is that points outside the perimeter are considered to be inside the search area. However, if the maximal distance is small enough, these points are close enough to the search area perimeter to be considered safe. Figure 82 illustrates such method.

Figure 82 shows an example of search area as interpreted by the Object Tracking behavior. The size of the gaps and sectors outside the search area has been exaggerated in order to better illustrate the concept. As seen, all the points within the search cells or within a minimum distance of two cells are considered to be part of the area.

The pseudo-code of function 14 describes the method to verify if the UAV or object is inside the search area.

```
Function 14:    isInsideSearchArea( areaCells[], position, minDist )

     for each cell areaCells[i]
          for each vertex v[j] of areaCells[i]
               if v[j].x = position.x
                    v[j].x ← v[j].x + ε
               end if
          end for
          if number of times which ray from position to x+
             intercepts perimeter of areaCells[i] is odd
               return true;
          end if
     end for
     Define integer closeCells = 0;
     for each cell areaCells[i]
          if distance between position and one or more edges of
             perimeter of areaCells[i] is smaller than minDist
               closeCells ← closeCells + 1;
          end if
     end for
     if closeCells ≥ 2
          return true;
     end if
     else
          return false;
     end else
```

6.4.1.2 Find Object

When entering the Object Tracking state, Find Object state is activated. Before entering, the object is spotted the first time. However, since the spotting could be result of noisy interference (hence a false detection), the object is not considered to be found yet. The objective of the system in this state is to optimize the probability of spotting the object again.

Therefore, the UAV is commanded to move quickly towards the position of the first spot. The UAV continues to move in this direction even when surpassing the first spot position. If the object is considered lost, the system leaves the sub-state and the Object Tracking state. If the object is spotted again, it is considered to be found and the system transits into Perform Tracking sub-state.

The flowchart in figure 83 illustrates the processes and features executed by the Object Tracking Behavior object when Find Object sub-state is active.

6.4.1.3 Perform Tracking

When entering this sub-state, the Object Tracking behavior considers the object to be found. Therefore, the Object Tracking behavior's filter is applied to estimate the true position of the object based on data from the VC, to predict the current position of the object when the VC provides no data and to predict future positions of the object. In this state, the pursuit strategy is applied.

If the object is considered lost or the object or UAV have left the search area perimeter, the system leaves to Ascend sub-state. The system remains in the Perform Tracking sub-state while the object is considered to be tracked unless commanded by the operator to cease the tracking or if the link with GCS is lost.

The flowchart in figure 84 illustrates the processes and features executed by the Object Tracking Behavior object when Perform Tracking sub-state is active. In the flowchart, the pursuit strategy is applied as shown in the flowchart presented on subsection **6.3.2.3**.

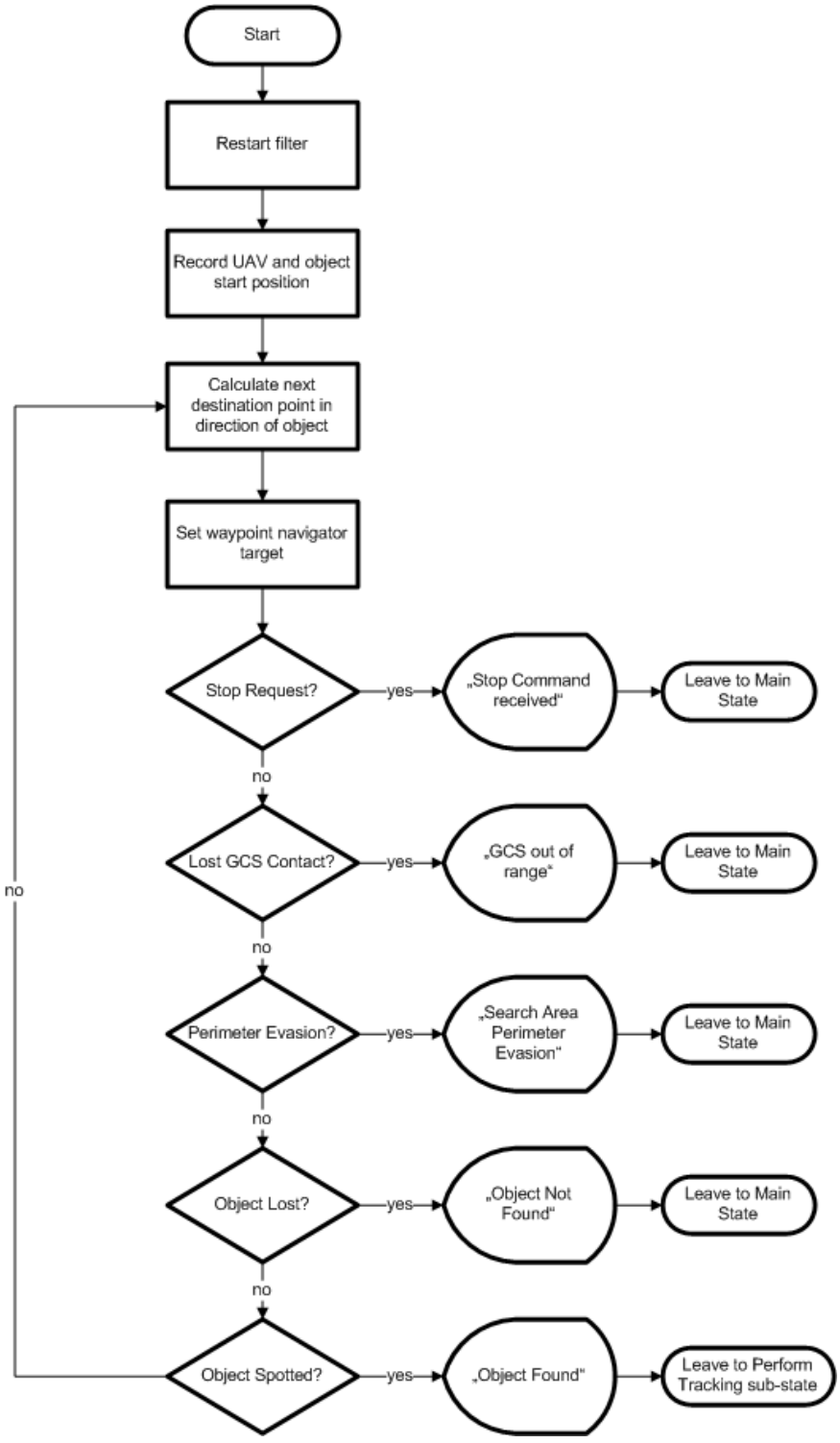

Figure 83: Flowchart description of Find Object sub-state

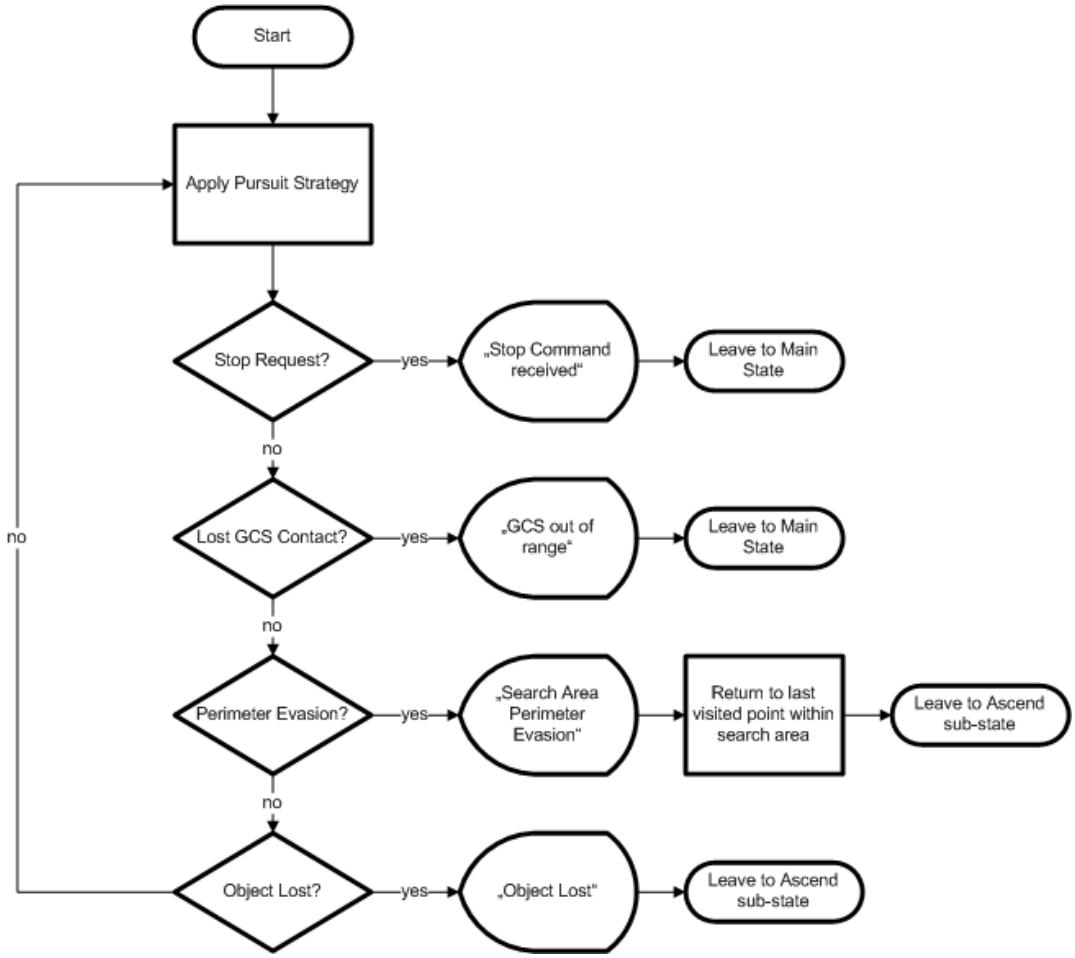

Figure 84: Flowchart description of Perform Tracking sub-state

6.4.1.4 Ascend

The system transits into this sub-state when the tracking is considered to have failed, i.e. when the object is lost or when a perimeter evasion occurs. In case the UAV has fled perimeter, the Object Tracking behavior object uses the waypoint navigator to command the aircraft to return to the last visited point within the boundaries of the search area. The behavior object also uses the navigator to command the helicopter to ascend, providing a broader camera view, hence enhancing the probability of re-finding the object.

The Ascend sub-state is left when the time elapsed since the last spotting of the object has overcome a threshold. This threshold is pre-defined and configurable. If the object is once again spotted, the system re-enters Find Object sub-state, resetting the behavior object's filter and restarting the Object Tracking task.

The flowchart in figure 85 illustrates the processes and features executed by the Object Tracking Behavior object when Ascend sub-state is active.

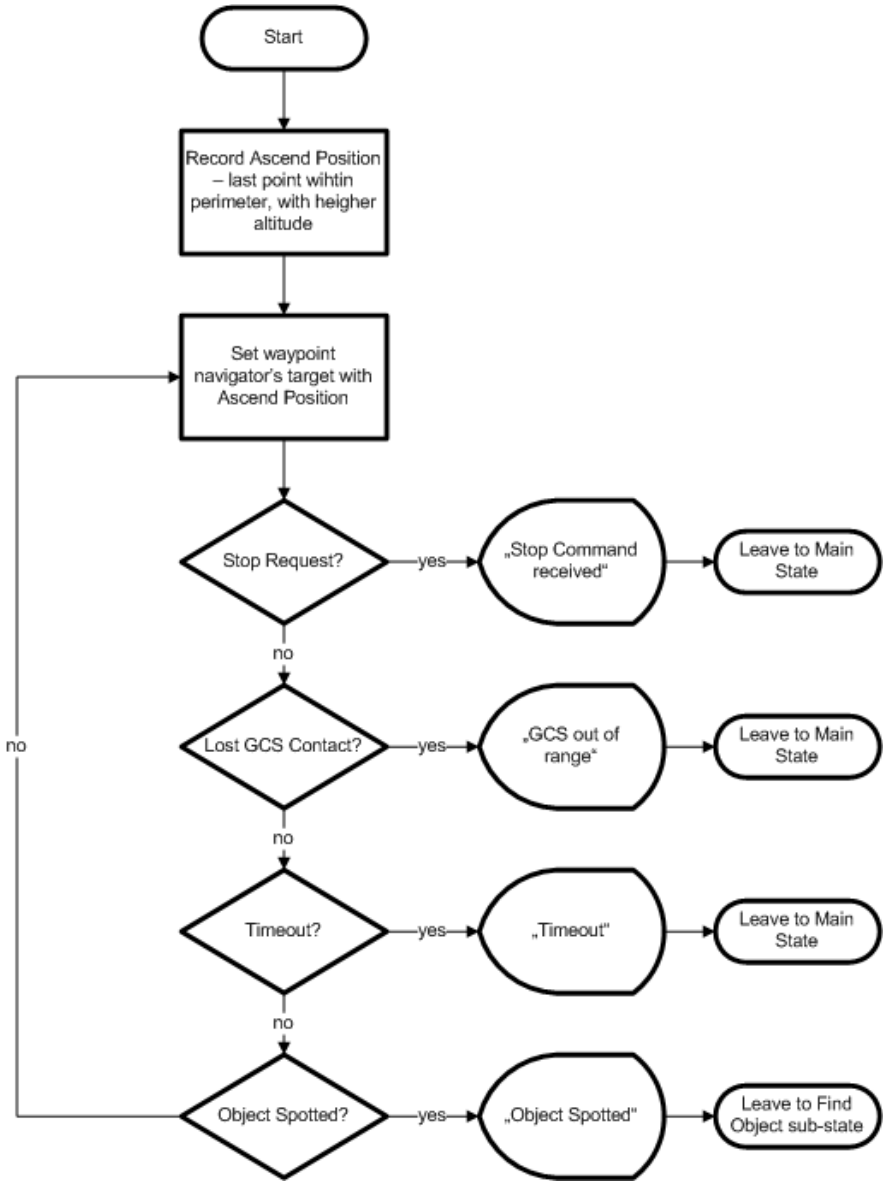

Figure 85: Flowchart description of Ascend sub-state

6.5 Results

This section presents the practical results obtained with the implementation of the Object Tracking behavior, as well as the analysis of such outcome. The features designed to address the Tracking sub-problem have been evaluated in tests regarding the performance of the Kalman filters when estimating the target's position and tests concerning the performance of the target pursuit strategy. The features designed to address the Management sub-problem have been evaluated in tests regarding the capability of the behavior to recognize if the UAV

is within the boundaries of the search area and in tests concerning the management of events and temporal stages of the behavior.

It is important to remember that, unless explicitly remarked otherwise, the coordinates shown in this section are conformant to the NED reference system, i.e. (North (m), East (m), Down (m)) set at ground level.

6.5.1 Target Pursuit

In this subsection we present the results of tests concerning the Object behavior's pursuit strategy presented in subsection **6.3.2**. HITL simulation was used in the tests, along with the target detection system, which includes the camera and the pattern detection software. The tests were designed both to validate the behavior's object tracking capabilities and to verify the performance enhancement when compared to ARTIS' previous object tracking abilities.

The scenario proposed was tracking the ground object in an obstacle-free space, without search area restrictions and maintaining constant tracking height. The target was manually driven, maneuvering with the intention of testing the performance of the tracker, e.g. suddenly changing the direction of movement, moving at full speed and performing sharp turns. Also, the capability of the Object Tracking behavior to track the object when it is off the camera view was tested. Three series of tests were executed, one using filter PV, one using filter PVA and one using ARTIS' previous tracker.

In the tests using the Object Tracking behavior, its parameters were tuned to enhance the performance. These parameters include those of the filters (i.e. measurement and process noise covariance) and those of the pursuit strategy (i.e. the time ahead for target position prediction in approach and accompany modes and the distance between UAV and object which constitutes the threshold between accompany and approach modes as shown in figure 77). The full set of parameters is shown in table 13. The results of the tests are presented in figures 86, 87 and 88.

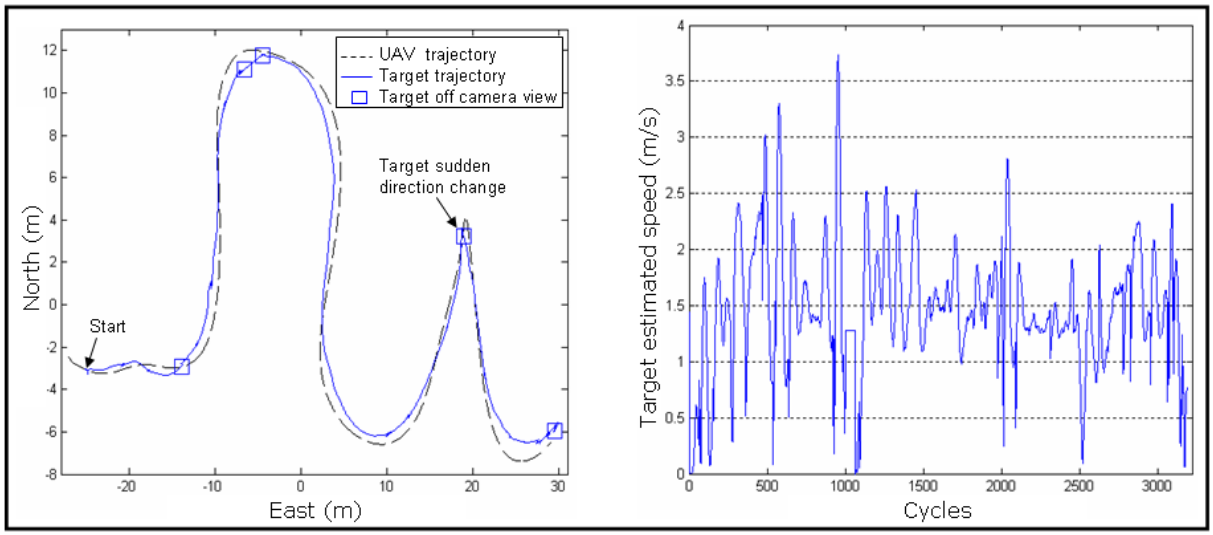

Figure 86: Target pursuit test, first setting. The Object Tracking behavior with filter PVA was used. The right graph presents the trajectory of the UAV and the estimated trajectory of the target. The left figure presents the target's estimated speed [m/s] throughout the pursuit.

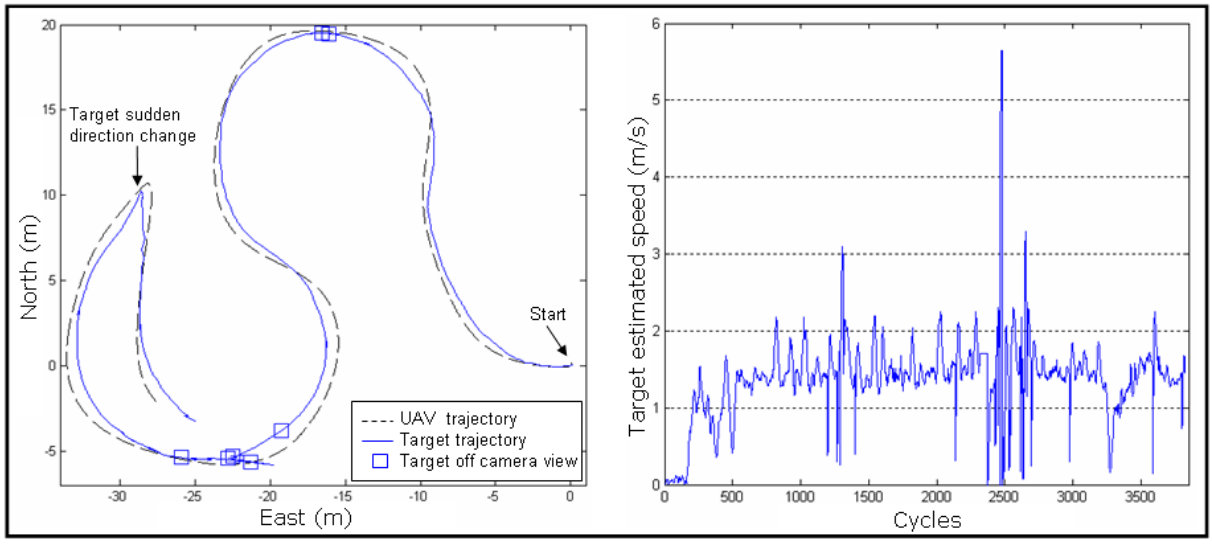

Figure 87: Target pursuit test, second setting. The Object Tracking behavior with filter PV was used. The right graph presents the trajectory of the UAV and the estimated trajectory of the target. The left figure presents the target's estimated speed [m/s] throughout the pursuit.

As presented on figures 86, 87 and 88, in the tests presented the tracker versions were capable of maintaining the target within camera view. However, in the test with ARTIS' previous tracker, the tracking was not possible with the object at full speed. It was necessary to constantly slow down the object in order to allow the UAV to "catch up", resulting in an average target speed of 0.9 [m/s]. In the tests using the Object tracking behavior, the UAV was capable of performing the tracking with the object moving at full speed at all times. The estimated average target speed in the first test (with Object Tracking behavior and filter PVA)

and in the second test (with Object Tracking behavior and filter PV) were of, respectively, 1.5 [m/s] and 1.4 [m/s]. The improvement of target average speed when comparing with the previous tracker was of 67% in the first case and of 56% in the second.

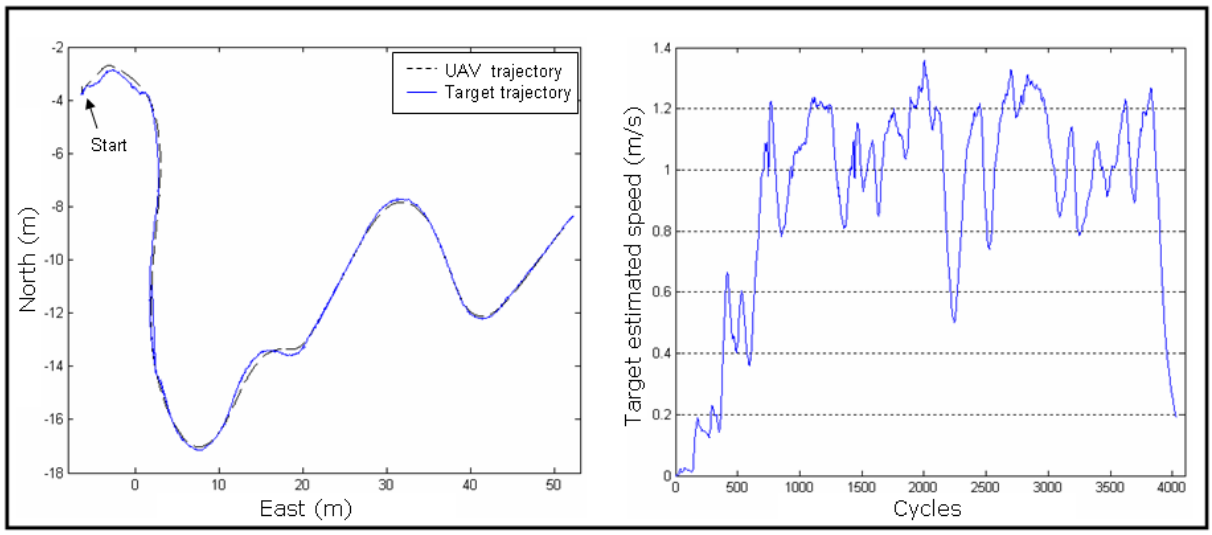

Figure 88: Target pursuit test, third setting. ARTIS' previous tracker was used. The right graph presents the trajectory of the UAV and the estimated trajectory of the target. The left figure presents the target's estimated speed [m/s] throughout the pursuit.

Table 13: Test parameters

Parameter	Value
UAV height	12 [m]
Measurement noise covariance	0.01
Process noise covariance	10
Distance threshold	5 [m]
Accompany prediction time	400 [ms]
Approach prediction time	100 [ms]

Another aspect in which the Object Tracking behavior showed improvement when compared to ARTIS' previous tracker regards the target speed at the beginning of the tracking. As shown in figure 88, in the third test it was necessary to keep the object initially with a slow speed (around 0.5 [m/s]) in order to allow the UAV to accelerate and "catch up" with the target. In the other tests, the Object Tracking behavior provided a quick and vigorous response, starting the tracking with the object at full speed.

The Object Tracking behavior also showed improvement in situations when some maneuvers of target resulted in moments when the object was not within camera view. In such moments, the previous tracker would command the UAV to fly to the position where the object was last seen, disregarding the target's motion. If the object would continue moving, the previous tracker would mostly not be able to re-find it. The Object Tracking behavior, however, used its capability of predicting the target's position to continue the pursuit in such moments. Figures 86 and 87 show moments when the object was out of sight for at least 100 [ms]. In the first test, the most critical object maneuver resulted in 1240 [ms] without spotting the object; in the second, it resulted in 1120 [ms] without a positive spot. In both situations, the UAV was able to re-find the object.

One particular maneuver evidenced another aspect in which the Object Tracking behavior provided superior performance. In this maneuver, which is shown in figures 86 and 87, the target suddenly changes the direction of its motion without stopping. This maneuver is particularly demanding, since the UAV has a much greater inertia than the object and the object can be easily lost, as shown in figure 60. In the first test (with the PVA filter), the UAV lost sight of the object for 1.24 [s], but through prediction of target trajectory the Object Tracking behavior enabled the aircraft to maintain the tracking and regain visual contact with the object *without* slowing the target down. In the second test (with the PV filter), the helicopter was capable of keeping track of the object while maintaining it within sight through the whole maneuver. ARTIS' previous tracker was also used to track such maneuver, but it failed to maintain the tracking in all trials.

The tests showed that the Object Tracking behavior has enhanced ARTIS' capability of pursuing a ground object, introducing the ability to predict the object's trajectory when it is obscured or out of camera view. Furthermore, the behavior provided a tracker which is more responsive to the object's motion than the previous version, being able to track a faster and more agile target. Thus, the Object Tracking behavior has successfully achieved its requirements regarding target tracking.

It is important to remark that such tests did not expose the extent of the capabilities of the Object Tracking behavior, since the demands of tests were limited by the target's maximum speed. In order to test the limits of the tracking ability provided by the behavior, it would be necessary to use a faster recognizable ground object.

6.5.2 Search Area Evasion

In this subsection we present the results of tests concerning the capabilities of the Object Tracking behavior of recognizing if the UAV is located within the boundaries of the search area. As stated in subsection **5.3.1**, the search area is divided into convex cells. Therefore, positions inside one of the convex cells or *between* cells (i.e. in region where the original search area is divided into two or more cells) must be recognized as part of the search area.

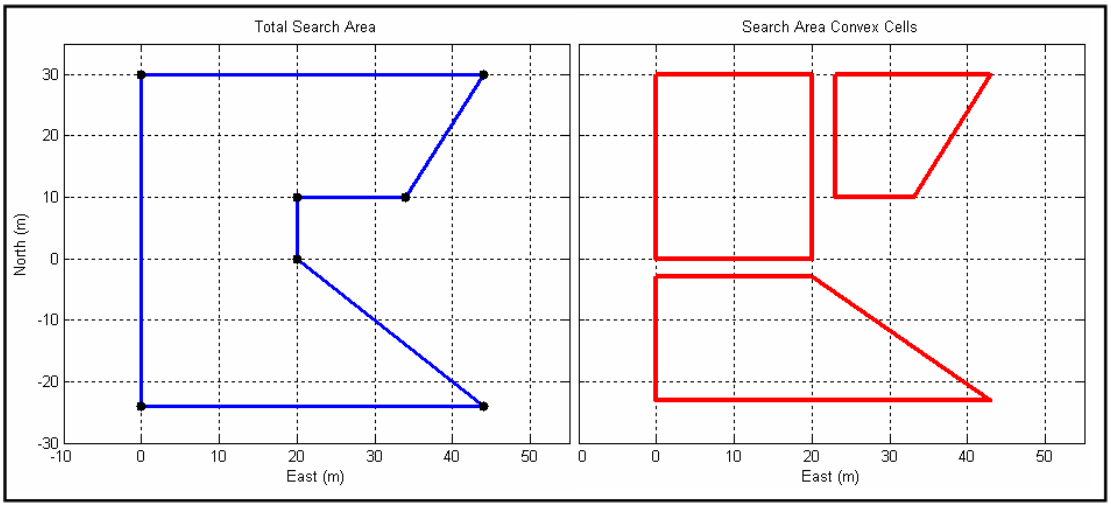

Figure 89: Search area and convex cells overview

Using Unit testing, a concave perimeter – the original search area – was divided into convex cells. Figure 89 displays the total search area and the respective convex cells. The tests consisted in using the Object Tracking behavior to check for several points if the position is within or outside the search area. The points tested are shown in figure 90.

In this test, the configuration of the system was such that the Object Tracking behavior considered search cells separated by a distance of 3 meters to be connected. Therefore, the cells **1** and **2**, as well as **1** and **3**, are considered connected. As presented, seven positions were tested. The points **A**, **B** and **C** are each inside one of the convex cells, while the points **D** and **E** are located between connected cells. Therefore, these points are inside the search area. Points **F** and **G**, however, are outside the search area. Using Unit testing, it was verified that the Object Tracking behavior was able to properly accuse which points were inside and which were outside the search area.

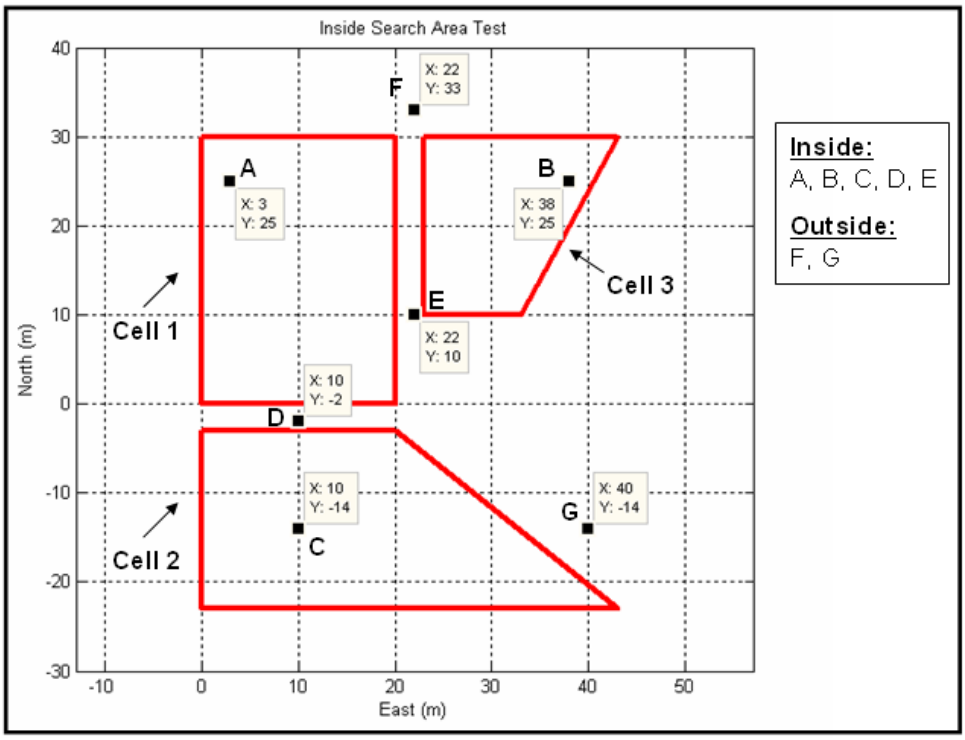

Figure 90: inside search area test

6.5.3 Behavior Management

Using the SITL simulation presented on section **5.5.2**, it was possible to determine if the Object Tracking behavior is capable of properly *online* managing the events and stages of the mission. The result of the simulation is presented once again in figure 91.

As stated before, from point 2 to point 3 the UAV pursued a target until leaving the perimeter at point 3. After the evasion of the search area, the Object Tracking behavior commands the aircraft to return to the perimeter's border (point 5) and to increase its position's height. Afterwards, the UAV waited for a certain period of time to check if the object would return to the search area. After the timeout, the system continued the original mission. Figure 92 presents a more detailed description of the temporal evolution of the test.

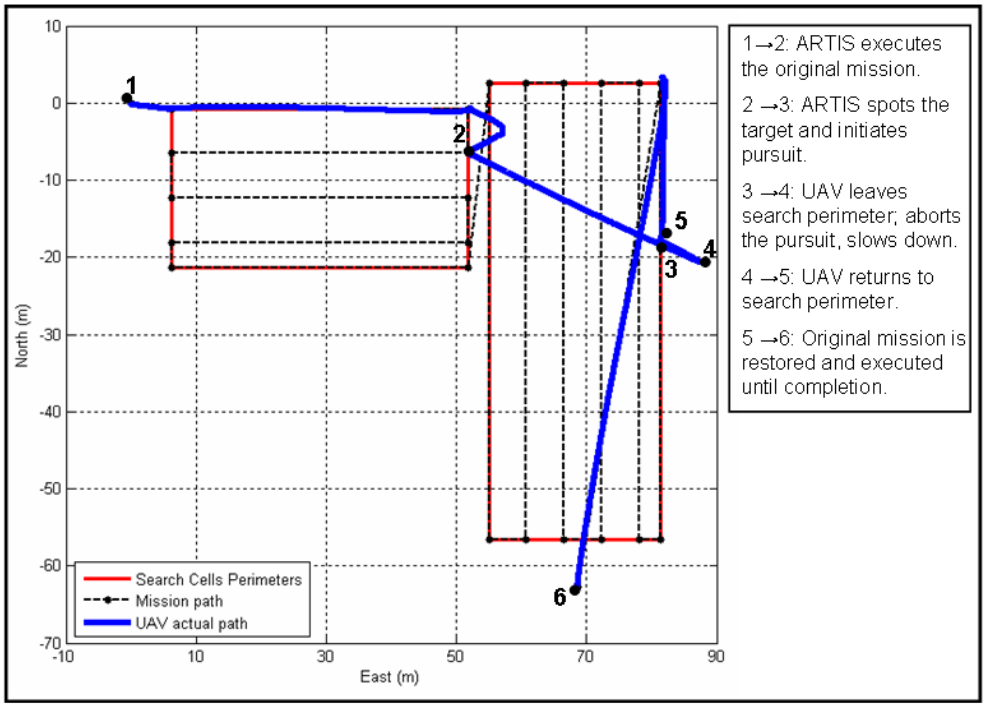

Figure 91: Management SITL test

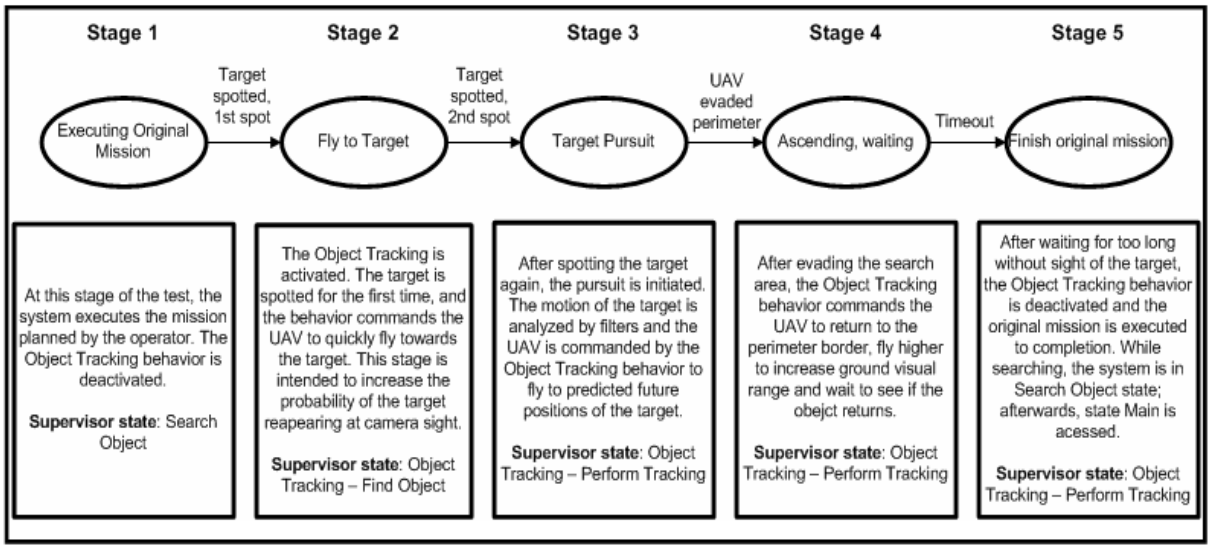

Figure 92: Temporal evolution of the management SITL test

The presented temporal evolution of the reset search task was checked using SITL simulation and corresponds to the expected functionality of the behavior, as shown on section **6.4**.

7 Future Work

The development of the three intelligent behaviors provided solutions to the problems proposed which reached the respective objectives and restraints. However, several aspects of the intelligent behaviors present room for improvement, especially with respect to efficiency. In this chapter, we present aspects of the intelligent behaviors which could be improved in the future.

7.1 Fly Home

As presented on chapter **4**, the mission planned by the Fly Home behavior describes the inverse path of the original mission. Such approach provides a safe route returning to the mission's starting point, and therefore reaching the behavior's objective. However, in certain situations it is possible to find alternative paths contained in the original path which would result into shorter missions. Such situations occur frequently when it is possible to find a "shortcut", as shown in figure 93.

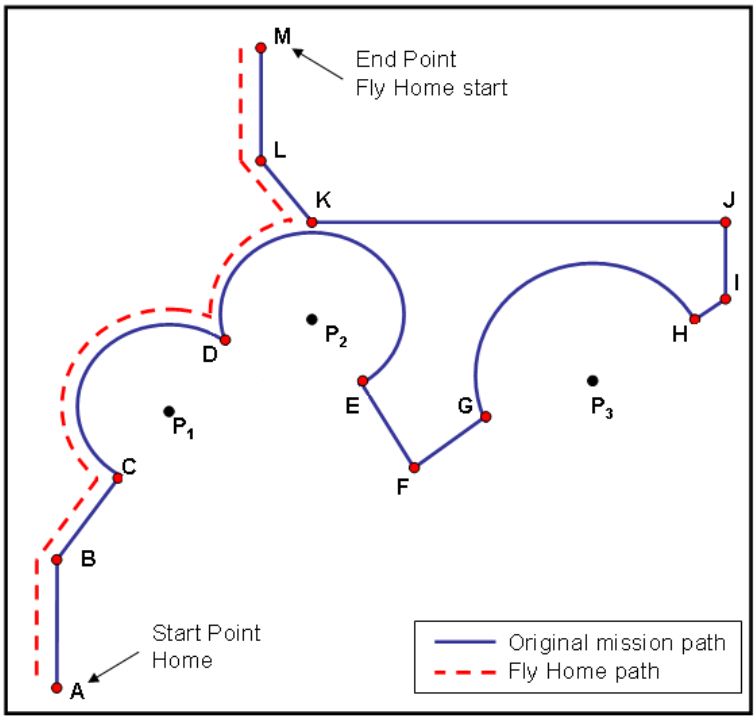

Figure 93: Fly Home mission with enhanced efficiency

As seen in figure 93, the path of the mission illustrated could be greatly reduced by producing a mission linking directly point *K* and the path connecting *D* and *E*. Such enhancement in efficiency can be achieved by checking if the distances between path segments of the Fly Home are within a minimum distance. When such condition is met, the path segments between the close segments (and the respective behaviors) are discarded.

Another aspect of the Fly Home mission which leaves room for improvement concerns the Fly To behavior. The Fly Home behavior currently replaces Fly To behaviors with Hover To behaviors. Such approach reduces the efficiency of the mission with respect to time duration, since the Fly To behavior provides a greater flight speed. Furthermore, some missions are planned such that the spline representing the Fly To trajectory "avoids" an obstacle in between waypoints. In such missions, the Fly Home behavior cannot calculate a safe path home. Therefore, a better solution would be inverting the Fly To's spline trajectory instead of substituting it with a Hover To.

7.2 Search Object

As shown on chapter **6**, the Search Object calculates the search area by computing its convex cells. The Object Tracking behavior, as shown in chapter **7**, considers points in between these cells to be part of the search area. The inconvenient of this approach is that points close to gaps between cells which are not part of the original search area can be considered to be inside the perimeter, as shown in figure 82. Therefore, it would be desirable if the Search Object behavior had the capability of "merging" the convex cells and recuperating the original search area perimeter as shown in figure 94.

As seen in figure 94, the computational problem of representing the area as a polygon when there is a "hole" in the area is overcome by representing the perimeter as a polygon which "starts" and "ends" in the same edge.

Such approach not only has the advantage mentioned previously, but also enhances the efficiency with which the Object Tracking behavior checks is the aircraft or the object are within the search area perimeter. This is justified by the fact that the algorithm used for the check is then applied only once for each area, instead several times for each cell of the areas.

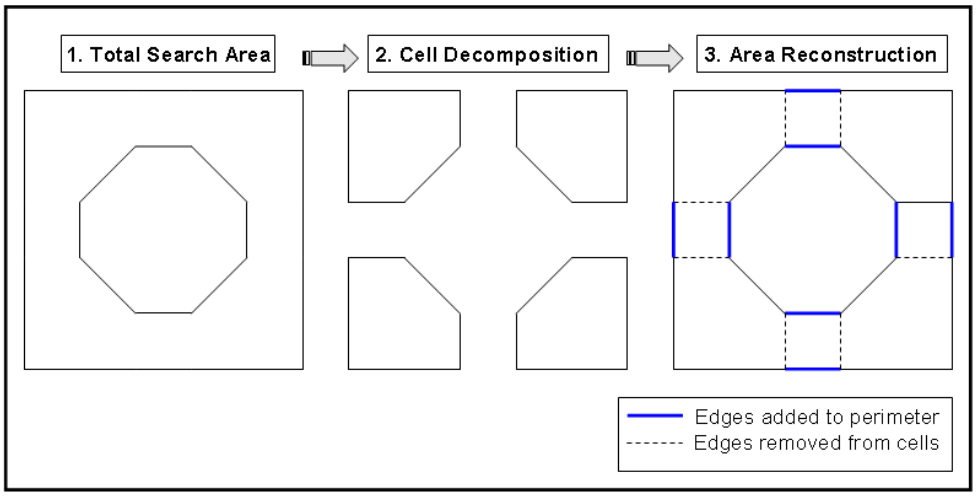

Figure 94: Search area reconstruction

7.3 Object Tracking

As shown in chapter **6**, the Object Tracking behavior uses a Kalman filter to estimate the position of the ground object and to predict its future position. As the motion dynamics of the object is unknown, the approach of using Interacting Multiple Model estimations would be able to provide better results, since it allows the use of different process models according to different "behaviors"[†] of the target [44]. Some target behaviors which could be described in different models include straight trajectory, curved trajectory, sudden changes of motion direction and target still.

Another aspect which presents room for improvement is the switching between Approach and Accompany pursuit modes. The criterion used does not take account of the target's motion direction. Figure 92 illustrates the issue.

The approach used at the moment for the pursuit strategy is using filters in x and y directions to command the helicopter's motion. However, as shown in figure 92, a strategy which would separate the filters in the direction of the object's velocity vector and in the perpendicular direction. This way, the strategy modes could be chosen and applied more efficiently.

As an example, in figure 95 the UAV is considered to be behind the target on y direction (activating Approach mode in y filter) and ahead on x direction (activating Accompany mode in x filter). However, this approach would result into a slower response on approaching the target in y' direction and a response too strong on x', causing the UAV to

[†] Relates to the target's several motion patterns, not to be confused with behavior-based paradigm

surpass the target in this direction. For these reasons, using different modes in the rotated reference directions *x'* and *y'* would provide a more efficient pursuit strategy.

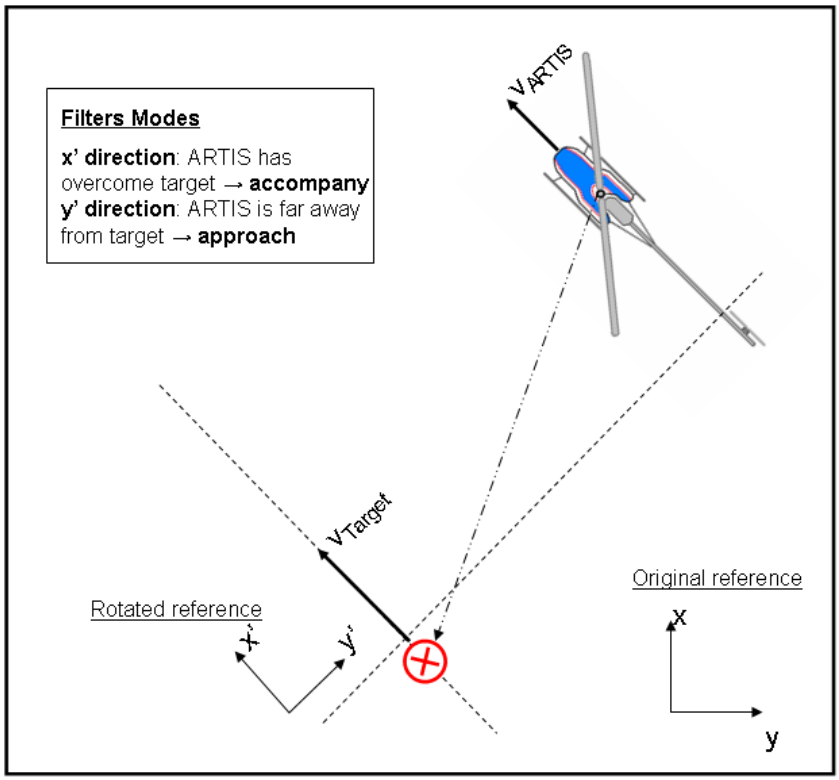

Figure 95: Pursuit mode switching with reference rotation

8 Conclusion

In this work, we presented solutions for the problem of developing intelligent behaviors for ARTIS VTOL. Such solutions provided ARTIS with enhanced decision capabilities, situational awareness and overall autonomy. The intelligent behavior modules designed achieved their objectives without the necessity of Path Planning and World Model modules, which are complex and "costly" from implementation and computational point of view.

The development of the Fly Home behavior provided the UAV with the capability of autonomously planning and executing a mission leading back to the starting point through a safe path. The Search Object behavior provided ARTIS with the capabilities of properly managing and restarting a search mission and recognizing the search area as planned by the operator using the Andrew Monotone Chain algorithm to calculate the area's convex cells. The Object Tracking behavior contributes with the enhancement of the object tracking capabilities of the onboard system. The behavior uses two types of Kalman filters to provide the estimation of the object's position. Furthermore, an algorithm based on the Jordan Curve Theorem was implemented to check if the UAV or the object is within the search area perimeter. Moreover, the coordinated application of the Search Object and Object Tracking behaviors provides ARTIS with awareness of the mission's objectives and autonomous management of every stage of a search and track task.

The behavior modules have been thoroughly tested using Unit testing and SITL and HITL simulation environments. The tests were designed to test the behaviors' efficiency and robustness in several different situations which ARTIS could face in a real mission scenario. Furthermore, the tests were used to tune parameters and optimize the behaviors' performance, e.g. the tuning of the Object Tracking behavior's filters and pursuit strategy mode switch criteria.

Finally, we proposed possible enhancements with regard to the behavior modules. Such improvements are intended to provide greater performance efficiency and robustness to the intelligent behaviors.

References

[1] DITTRICH, J. S.; THIELECKE, F.; SCHWANECK, H. Unmanned Rotorcraft Demonstrator ARTIS: Challenges in Autonomous Control and Teaming. In: AHS Annual Forum, 2004, Baltimore. **Proceedings…** Baltimore, 2004.

[2] HILL, R.; CHEN, J.; GRATCH, J.; ROSENBLOOM, P.; TAMBE, M. Intelligent agents for the synthetic battlefield: A company of rotary wing aircraft. **Innovative Applications of Artificial Intelligence**, 1997.

[3] PUTZER, H.; ONKEN, R. COSA A generic cognitive system architecture based on a cognitive model of human behavior. **Cognition, Technology and Work**, v. 5, 2003.

[4] KARIM, S.; HEINZE, C. Experiences with the design and implementation of an agent-based autonomous UAV controller. In: Fourth International Joint Conference on Autonomous Agents and Multiagent Systems, 2005. **Proceedings...** 2005.

[5] AGRE, P. E.; CHAPMAN, D. Pengi: an implementation of a theory of activity. **Computation & intelligence: collected readings**, American Association for Artificial Intelligence, p. 635–644, 1995.

[6] TOAL, D.; FLANAGAN, C.; JONES, C.; STRUNZ, B. Subsumption Architecture for the Control of Robots. **IMC-13**, Limerick, 1996.

[7] BROOKS, R. S. A robust layered control system for a mobile robot. **IEEE Journal of Robotics and Automation**, v. 2, n. 1, p.14–23, 1986.

[8] BROOKS, R. A. Integrated Systems Based on Behaviors. **SIGART Bulletin**, v. 2, n. 4, 1992.

[9] ADOLF, F.; THIELECKE, F. A Sequence Control System for Onboard Mission. In: Infotech Aerospace, 2007, Rohnert Park, CA. **Proceedings...** Rohnert Park, CA, 2007.

[10] PIRJANIAN, P. The Notion of Optimality in Behavior-Based Robotics. **Journal of Robotics and Autonomous Systems**, 1999.

[11] SARIPALLI, S.; NAFFIN, D. J.; SUKHATME, G. S. Autonomous Flying Vehicle Research at the University of Southern California. In: First International Workshop on Multi-Robot Systems, 2002. **Proceedings...** 2002.

[12] DOHERTY, P.; GÖSTA, G.; KUCHCINSKI, K.; SANDEWALL, E.; NORDBERG, K.; SKARMAN, E.; WIKLUND, J. The WITAS unmanned aerial vehicle project. In: European Conference on Artificial Intelligence, 2000. **Proceedings...** 2000.

[13] SCHRAGE, D.; VACHTSEVANOS, G. Software-Enabled Control for Intelligent UAVs. In: International Conference on Control Applications, 1999, Hawaii. **Proceedings...** Hawaii, 1999.

[14] BONASSO, R. P.; FIRBY, J.; GAT, E.; KORTENKAMP, D.; MILLER, D. P.; SLACK, M. G. Experiences with an Architecture for Intelligent, Reactive Agents. **Journal of Experimental & Theoretical Artificial Intelligence**, v. 9, n. 2/3, p. 237–256 , April 1997.

[15] SHIM, D.; CHUNG, H.; KIM, H. J.; SASTRY, S. Autonomous Exploration in Unknown Urban Environments for Unmanned Aerial Vehicles. In: AIAA GNC Conference, 2005, San Francisco, CA. **Proceedings...** San Francisco, CA, 2005.

[16] PETERSSON, P.; DOHERTY, P. Probabilistic Roadmap Based Path Planning for an Autonomous Unmanned Aerial Vehicle. In: AAAI Workshop on Connecting Planning Theory with Practice, ICAPS, 2004. **Proceedings...** 2004.

[17] HAREL, D.; Statecharts: A Visual Formalism for Complex Systems. **Sci. Comput. Programming**, v. 8, p. 231-274, 1987.

[18] DOHERTY, P.; HASLUM, P.; HEINTZ, F.; MERZ, T.; NYBLOM P.; PERSSON, T.; WINGMAN, B. A distributed architecture for autonomous unmanned aerial vehicle experimentation. In: 7th International Symposium on Distributed Autonomous Robotic Systems, 2004. **Proceedings...** 2004.

[19] BROOKS, R. A. Intelligence without Representation. **Artificial Intelligence**, 47, 139-159, 1991.

[20] BROOKS, R. A. Intelligence without Reason, In: 12th International Joint Conference on Artificial Intelligence. Menlo Park, CA. **Proceedings...** Menlo Park, CA, 1991.

[21] PRAXEDES, L. G. **Advanced Three-Dimensional Route Planning under Complex Constraints**. 2008. Trabalho de Conclusão de Curso. (Graduação) – Instituto Tecnológico de Aeronáutica, São José dos Campos.

[22] HUANG, H.-M.; MESSINA, E.; ALBUS, J. Toward a Generic Model for Autonomy Levels for Unmanned Systems (ALFUS). In: Performance Metrics for Intelligent Systems (PerMIS) Workshop, 2003, Gaithersburg, MD. **Proceedings...** Gaithersburg, MD, 2003.

[23] HUANG, H.; MESSINA, E.; ALBUS, J.; WADE, R.; ENGLISH, R. Autonomy Levels for Autonomous Vehicles: Preliminary Results. In: Proceedings of the SPIE Defense and Security Symposium, 2004 Orlando, FL. **Proceedings…** Orlando, FL, 2004.

[24] LANGER, A. **Three-Dimensional Path Planning for an Unmanned Rotorcraft Using Probabilistic Roadmaps**. 2007. Trabalho de Conclusão de Curso. (Graduação) – Instituto Tecnológico de Aeronáutica, São José dos Campos.

[25] ANDERT, F.; STRICKERT, G.; THIELECKE, F. Depth Image Processing for Obstacle Avoidance of an Autonomous VTOL UAV. In: Deutscher Luft- und Raumfahrtkongress, 2006, Germany. **Proceedings...** Germany, 2006.

[26] IEEE Standards Board, IEEE Standard for Software Unit Testing: An American National Standard. **IEEE Standards: Software Engineering**, **Volume Two: Process Standards**, 1999.

[27] BERNATZ, A.; THIELECKE, F. Navigation of a Low Flying VTOL Aircraft With the Help of a Downwards Pointing Camera. In: AIAA Guidance, Navigation, and Control Conference, 2004, Providence, Rhode Island. **Proceedings...** Providence, Rhode Island, 2004.

[28] DE MELO PONTES E SILVA, L. **Mission and Task Management for a Group of Autonomous Agents**. 2007. Trabalho de Conclusão de Curso. (Graduação) – Instituto Tecnológico de Aeronáutica, São José dos Campos.

[29] ANDREW, A. M. Another efficient algorithm for convex hulls in two dimensions. **Information Processing Letters**, v. 9, n. 5, p. 216–219, 1979.

[30] KIRKPATRICK, D.; SEIDEL, R. The ultimate planar convex hull algorithm? **SIAM Journal on Computing**, v. 15, p. 287-299, 1986.

[31] PREPARATA, F. P., AND HONG, S. J. Convex hulls of finite sets of points in two and three dimensions. **Communications of the ACM**, v. 20, p. 87-93, 1977.

[32] JARVIS, R. A. On the identification of the convex hull of a finite set of points in the plane. **Information Processing Letters**, v. 2, p. 18–21, 1973.

[33] GRAHAM, R.L. An efficient algorithm for determining the convex hulls of a finite point set. **Information Processing Letters**, v. 1, p.132-133, 1972.

[34] BARBER, C. B.; DOBKIN, D. P.; HUHDANPAA, H. The quickhull algorithm for convex hulls. **ACM Transactions on Mathematical Software**, v.22, n.4, p. 469–483, 1996.

[35] KALLAY M. The complexity of incremental convex hull algorithms. **Information Processing Letters**, v. 19, p. 197, 1984.

[36] AVIS, D.; BREMNER, D. How good are convex hull algorithms? In: 11th Annual ACM Symposium on Computational Geometry, 1995. **Proceedings...** 1995.

[37] TOR, S. B.; MIDDLEDITCH A. E. Convex decomposition of simple polygons. **ACM Transactions on Graphics**, v.3, n. 4, p. 244-265, 1984.

[38] AVERBUCH, A.; ITZIKOWITZ, S.; KAPON, T. Radar target tracking – viterbi versus IMM. **IEEE Transactions on Aerospace and Electronic Systems**, v. 27, n.3, p. 550–563, 1991.

[39] KIRUBARAJAN, T.; BAR-SHALOM, Y.; PATTIPATI, K. R.; KADAR, I. Ground Target Tracking with Topography-Based Variable Structure IMM Estimator. **IEEE Transactions on Aerospace and Electronic Systems**, v. 36, n.1, p.26–46, 2000.

[40] FOX, D.; HIGHTOWER, J.; LIAO, L.; SCHULZ, D.; BORIELLO, G. Bayesian filtering for location estimation. **IEEE Pervasive Computing**, v. 2, n. 3, p. 24–33, 2003.

[41] WELCH, G.; BISHOP, G. **An Introduction to the Kalman Filter**. University of North Carolina at Chapel Hill, 2001. Disponível em: <http://www.ece.stevens-tech.edu/~ymeng/courses/robotics/papers/kalman.pdf >. Acesso em 19 dez. 2007.

[42] NUMMIARO, K.; KOLLER-MEIER, E.; VAN GOOL, L. An Adaptive Color-Based Particle Filter. **Image and Vision Computing**, v. 21, n. 1, p. 99-110, 2003.

[43] MAZOR, E.; AVERBUCH, A.; BAR-SHALOM, Y.; DAYAN, J. Interacting multiple model methods in target tracking: A survey. **IEEE Transactions on Aerospace and Electronic Systems**, v. 34, p. 103–123, 1998.

[44] KIRUBARAJAN, T.; BAR-SHALOM, Y. Kalman Filter vs. IMM Estimator: When We Need the Latter? In: SPIE Conference on Signal and Data Processing of Small Targets, 2000, Orlando, FL. **Proceedings...** Orlando, FL, 2000.

[45] LI, X. R.; JILKOV, V. P. A survey of maneuvering target tracking—Part V: Multiple-model methods. In: SPIE Conference on Signal and Data Processing of Small Targets, 2003, San Diego, CA. **Proceedings...** San Diego, CA, 2003.

[46] MILLER, G.; CLIFF, D. **Co-evolution of pursuit and evasion i: Biological and game-theoretic foundations**. School of Cognitive and Computing Sciences, University of Sussex, 1994. Disponível em: <ftp://ftp.cogs.susx.ac.uk/pub/reports/csrp/csrp311.ps.Z.>. Acesso em 19 dez. 2007.

[47] VIDAL, R.; SHAKERNIA, O.; KIM, H.; SHIM, D.; SASTRY, S. Probabilistic pursuit-evasion games: Theory, implementation and experimental evaluation. **IEEE Transactions on Robotics and Automation,** v. 18, n. 5, p. 662–669, 2002.

[48] MURRIETA-CID, R.; SARMIENTO, A.; HUTCHINSON, S. On the Existence of a Strategy to Maintain a Moving Target within the Sensing Range of an Observer Reacting with Delay. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, 2003. **Proceedings…** 2003.

[49] HELBLE, H.; CAMERON, S. 3-D Path Planning and Target Trajectory Prediction for the Oxford Aerial Tracking System. In: IEEE International Conference on Robotics and Automation, 2007. **Proceedings…** 2007.

[50] KIM, H. J.; SHIM, D. H. A flight control system for aerial robots: Algorithms and experiments. **Control Engineering Practice**, v. 11, p. 1389–1400, 2003.

[51] PREPARATA, F. P.; SHAMOS, M. I. **Computational Geometry**. Springer-Verlag, New York, p. 41-67, 1985.

[52] GLASSNER, A. S. **An Introduction to Ray Tracing.** Academic Press, p. 53-59, 1989.

Appendix A – Distance from Point to Segment

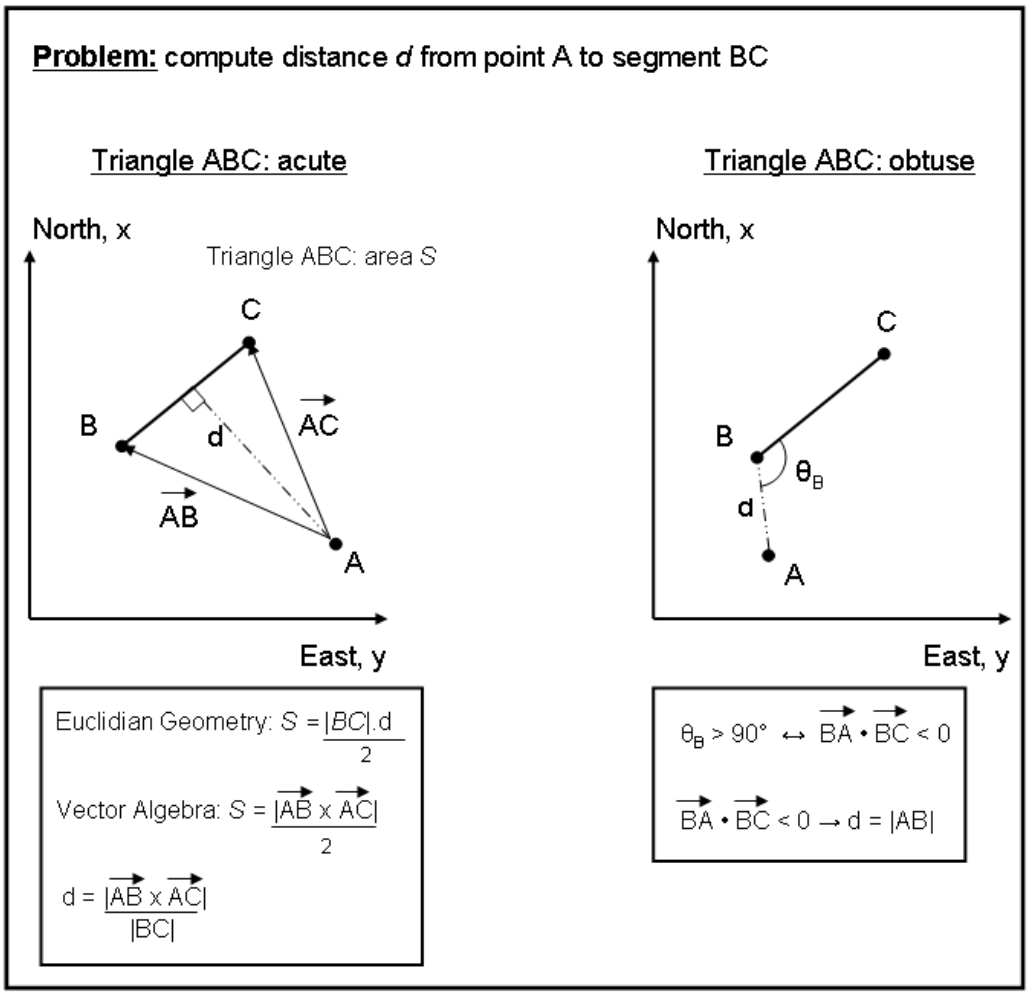

Figure 96: Distance from point to segment

Appendix B – Closest point of a straight segment to a test point

Formula 2:

Let $A=(x_a,y_a),\ B=(x_b,y_b),\ C=(x_c,y_c),$ such that triangle ABC is acute;

Let $m=\dfrac{y_c-y_b}{x_c-x_b},\ a=\dfrac{x_cy_b-x_by_c}{x_c-x_b},\ BC=\{(x,y)\in(x_b,y_b)\times(x_c,y_c)\ \big|\ y=mx+a\ \};$

Let $P=(x_p,y_p)\in BC\ \Big|\ \forall Q\in BC,\ |\overrightarrow{AQ}|\ \le\ |\overrightarrow{AP}|\ ;$

$P\in BC\Rightarrow y_p=mx_p+a$ (I);

Let $Q=(x,y)\in BC,\ d=\ |\overrightarrow{AQ}|\ \ \therefore\ \ d^2=(x-x_a)^2+(y-y_a)^2;$

d^2 is minimum for $Q=P$. Hence:

$$\left.\frac{\partial d^2}{\partial x}\right|_{\substack{x=x_p\\y=y_p}}=0\ \ \therefore\ \ \left.\frac{\partial}{\partial x}(x-x_a)^2+(y_p-y_a)^2\right|_{\substack{x=x_p\\y=y_p}}=0\ \ \therefore\ \ 2(x_p-x_a)+2(y_p-y_a)\frac{\mathrm{d}y}{\mathrm{d}x}=0\ \ \therefore$$

$$\therefore\ \ 2(x_p-x_a)+2m(y_p-y_a)=0\ \ \therefore\ \ (x_p-x_a)+\frac{y_c-y_b}{x_c-x_b}(y_p-y_a)=0\ \ \therefore$$

$$\therefore\ \ (x_p-x_a)(x_c-x_b)+(y_p-y_a)(y_c-y_b)=0\Rightarrow\overrightarrow{AP}\cdot\overrightarrow{BC}=0\Rightarrow\overrightarrow{AP}\perp\overrightarrow{BC};$$

Triangle ABC is acute $\Rightarrow\ \exists P\in BC\ |\ \overrightarrow{AP}\perp\overrightarrow{BC};$

$$(x_p-x_a)(x_c-x_b)+(y_p-y_a)(y_c-y_b)=0\ \overset{\text{(I)}}{\Rightarrow}\ (x_p-x_a)(x_c-x_b)+(mx_p+a-y_a)(y_c-y_b)=0\ \ \therefore$$

$$\therefore\ \ x_p[(x_c-x_b)+m(y_c-y_b)]=x_a(x_c-x_b)+(y_a-a)(y_c-y_b)\ \ \therefore$$

$$\therefore\ \ x_p=\frac{x_a(x_c-x_b)+(y_a-a)(y_c-y_b)}{(x_c-x_b)+m(y_c-y_b)}$$

$$y_p=mx_p+a\Rightarrow P=\left(\frac{x_a(x_c-x_b)+(y_a-a)(y_c-y_b)}{(x_c-x_b)+m(y_c-y_b)},m\cdot\frac{x_a(x_c-x_b)+(y_a-a)(y_c-y_b)}{(x_c-x_b)+m(y_c-y_b)}+a\right)$$

Appendix C – Andrew's Monotone Chain Algorithm

```
Algorithm: Andrew's Monotone Chain†

     Let S = {P = (P.x,P.y)} of N points
     Sort S by increasing x- and then y-coordinate.
     Let P[] be the sorted array of N points.

     Find points with 1st x min or max and 2nd y min or max:
     minmin = index of P with min x first and min y second
     minmax = index of P with min x first and max y second
     maxmin = index of P with max x first and min y second
     maxmax = index of P with max x first and max y second

     Compute the lower hull stack as follows:
     (1) Let L_min be the lower line joining P[minmin] with
     P[maxmin].
     (2) Push P[minmin] onto the stack.
     (3) Pushing the hull points:
     for each point P[i] between P[minmin] and P[maxmin]
          if (P[i] is above or on L_min)
               Ignore it and continue.
          end if
          while (there are at least 2 points on the stack)
          Let P_T1 = the top point on the stack.
               Let P_T2 = the second point on the stack.
               if (P[i] is left of the line from P_T2 to P_T1)
                    break out of this while loop.
          end if
               Pop the top point P_T1 off the stack.
     end while
     Push P[i] onto the stack.
     end for
```

† Algorithm description based on pseudo-code provided by http://www.softsurfer.com/Archive/algorithm_0109/algorithm_0109.htm

(4) Push *P[maxmin]* onto the stack.

Similarly, compute the upper hull stack.

Let $\boldsymbol{\Omega}$ = the join of the lower and upper hulls.

$\boldsymbol{\Omega}$ is the convex hull of **S**.

Appendix D – Waypoint List of Fly Home Test 1

Waypoint List – subsection **3.5.1**

Take Off
Hover To (-20, -10, -4, 0°)
Wait For (5)
Hover To (-10, -10, -4, 0°)
Hover To (-10, -10, -4, 45°)
Pirouette Flight (0, 0, -5, 10 [°/s])
Waypoint List Off
Hover To (0, 0, -5, 0°)
Hover To (10, 0, -6, 0°)
Hover To (10, 10, -6, 90°)
Hover Turn (20°, 10 [°/s])
Hover To (20, 20, -7, 45°)
Hover To (20, 20, -7, 90°)
Pirouette Around XY (20, 30, 10, 90°)
Hover Turn (-90°, 10 [°/s])
Pirouette Around XY (30, 40, 10, 180°)
Hover To (20, 50, -6, 180°)
Hover To (10, 60, -6, 135°)
Hover To (10, 70, -6, 90°)
Hover To (20, 70, -6, 0°)
Landing

Appendix E – Waypoint List of Fly Home Test 2

Waypoint list – subsection **3.5.2**

Take Off
Hover To (0, 0, -6, 0°)
Hover To (5, 0, -6, 0°)
Hover To (5, 0, -6, 90°)
Hover To (5, 10, -6, 90°)
Hover To (5, 10, -6, 180°)
Hover To (-10, 10, -6, 180°)
Pirouette Around XY (-10, 0, 5, -180°)
Hover To (-10, -10, -6, 26.6°)
Hover To (0, -15, -6, 26.6°)

FOLHA DE REGISTRO DO DOCUMENTO

1. CLASSIFICAÇÃO/TIPO	2. DATA	3. REGISTRO N°	4. N° DE PÁGINAS
TC	24 de outubro de 2008	CTA/ITA/TC-009/2008	167

5. TÍTULO E SUBTÍTULO:

Behavior-based High Level Control for VTOL UAV ARTIS

6. AUTOR(ES):

Maurício Moraes Carneiro

7. INSTITUIÇÃO(ÕES)/ÓRGÃO(S) INTERNO(S)/DIVISÃO(ÕES):

Instituto Tecnológico de Aeronáutica - ITA

8. PALAVRAS-CHAVE SUGERIDAS PELO AUTOR:

Behavior-based Control; Intelligent Behaviors; UAV High Level Control

9.PALAVRAS-CHAVE RESULTANTES DE INDEXAÇÃO:

Controle de aeronaves; Aeronave não-tripulada; Navegação autônoma; Planejamento de trajetória; Aeronaves de decolagem vertical; Helicópteros; Sistemas de computadores embarcados; Engenharia aeronáutica

10. APRESENTAÇÃO: **X Nacional** **Internacional**

ITA, São José dos Campos. Curso de Graduação em Engenharia Eletrônica. Orientador: Prof. Dr. Karl Heinz Kienitz; co-orientador: Dipl.Inf.(FH), M.Sc. Florian-Michael Adolf. Publicado em 2008.

11. RESUMO:

This work addresses the problem of providing the on-board system of an autonomous unmanned aerial vehicle (UAV) with the capabilities to plan and conduct complex missions with reduced off-board assistance. Such capabilities are referred to in this work as *Intelligent Behaviors*. In this work, three intelligent behaviors are proposed: *Fly Home*, *Search Object* and *Object Tracking*. The Fly Home intelligent behavior is intended to provide the UAV's on-board system with the capability of autonomously returning to the starting point of a given mission through a safe path. The behavior was designed considering the absence of online path planning and world model modules. The proposed approach is reorganizing, adapting and performing the tasks previously executed by the UAV so that the vehicle can return through the original path. The Search Object intelligent behavior is intended to manage a search mission planned offline, recognizing the areas in which the UAV is allowed to search and managing properly possible interruptions. To recognize the search areas, the total area is divided into convex cells and the Monotone Chain algorithm is used to calculate the perimeter of each individual convex cell. The Object Tracking intelligent behavior is intended to provide the capability of pursuing a moving ground object with the assistance of data provided by a visual recognition system. The motion characteristics of the ground object are estimated using a Kalman Filter. This work was developed with the ARTIS project at the German Aerospace Center (DLR), which aims to develop systems to autonomously operate rotorcraft vehicles.

12. GRAU DE SIGILO:

(X) OSTENSIVO **( ) RESERVADO** **( ) CONFIDENCIAL** **( ) SECRETO**